Алгоритмы и структуры данных

Сортировки

CS Center, Новосибирск

Алгоритм сортировки

Вход:

- последовательность ключей (k_1, k_2 … k_N), $k_i \in K$

Алгоритм сортировки

Вход:

- последовательность ключей (k_1, k_2 … k_N), $k_i \in K$

- Ключи можно сравнивать за O(1)

Алгоритм сортировки

Вход:

- последовательность ключей (k_1, k_2 … k_N), $k_i \in K$

- Ключи можно сравнивать за O(1)

Выход:

- новая последовательность ($k_{p[1]}$, $k_{p[2]}$ … $k_{p[N]}$), $k_{p[i]} \leq k_{p[i+1]}$

Алгоритм сортировки

Вход:

- последовательность ключей (k_1, k_2 … k_N), $k_i \in K$

- Ключи можно сравнивать за O(1)

Выход:

- новая последовательность ($k_{p[1]}$, $k_{p[2]}$ … $k_{p[N]}$), $k_{p[i]} \leq k_{p[i+1]}$
- иная постановка: найти перестановку индексов p[1], p[2] … p[N]

Алгоритм сортировки: свойства

Свойства, которыми может обладать

- In-place
- Stable

Пример: сортировка выбором

- N итераций {
  - Находим минимальный элемент
  - Исключаем его из последовательности
  - Добавлеям его в конец ответа

  }

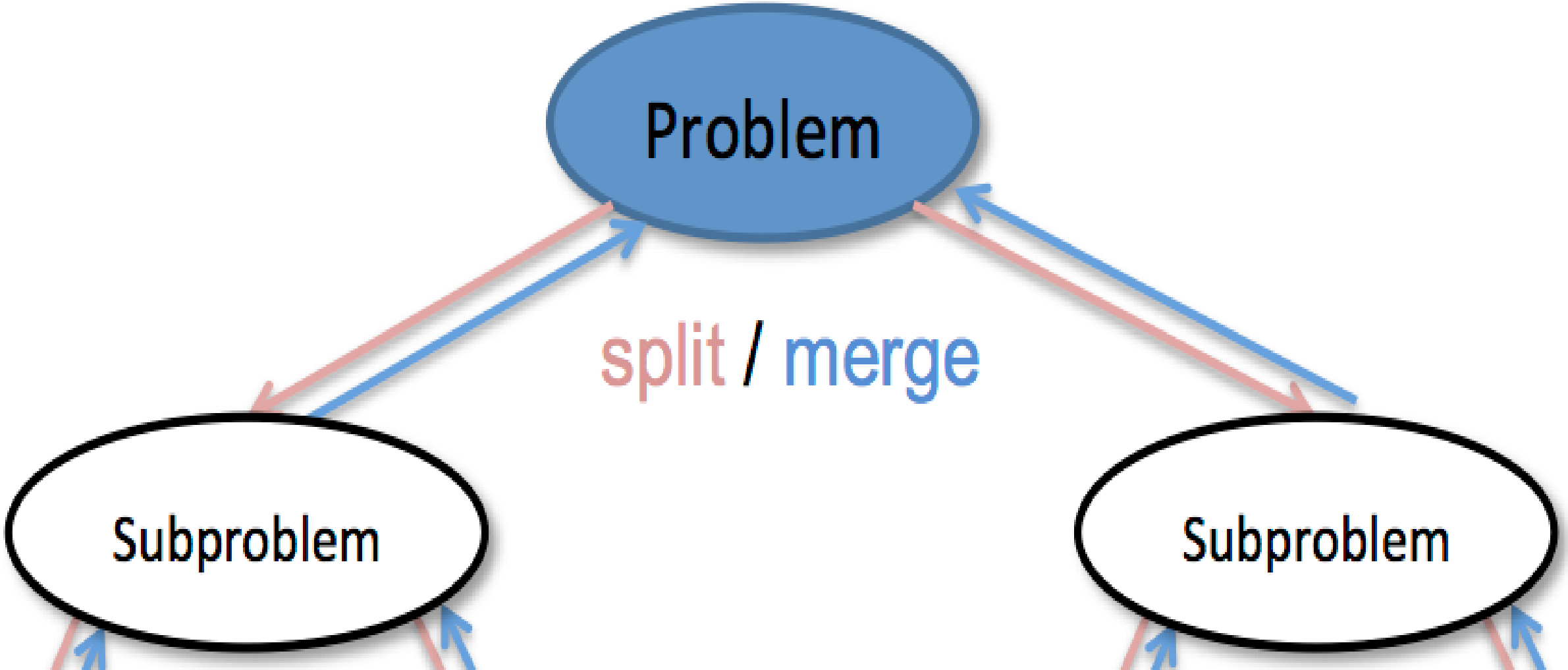
Пример: сортировка выбором

- N итераций {
  - Находим минимальный элемент
  - Исключаем его из последовательности
  - Добавлеям его в конец ответа

  }

Время работы?
In-place?
Stable?

Общая идея: разделяй и властвуй

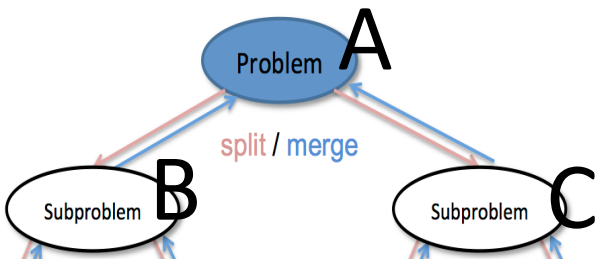


Разделяй и властвуй: алгоритмы сортировки

- QuickSort
- MergeSort

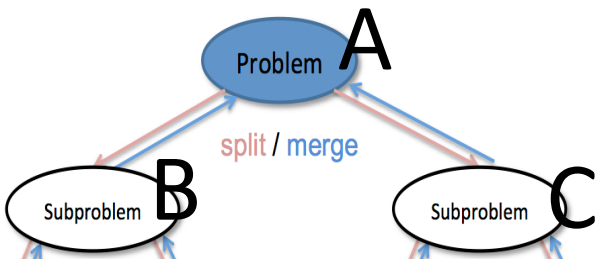
Quick sort

Применяем идею «Разделяй и властвуй»



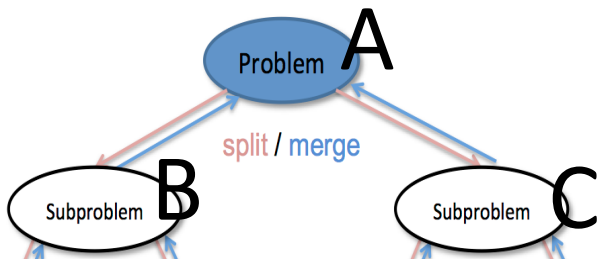
Quick sort

1. Применяем идею «Разделяй и властвуй»
2. ???????
3. PROFIT!



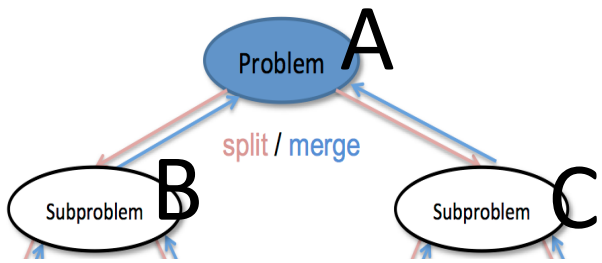
Quick sort

1. Из массива A делаем два подмассива B и C



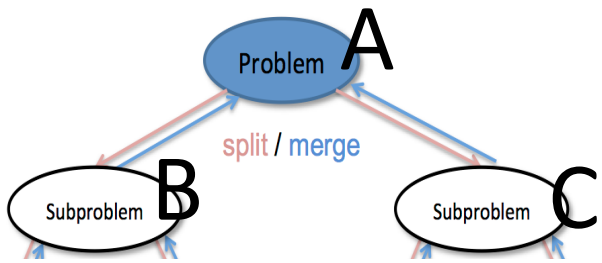
Quick sort

1. Из массива A делаем два подмассива B и C
   - Выбираем *pivot* – опорный элемент



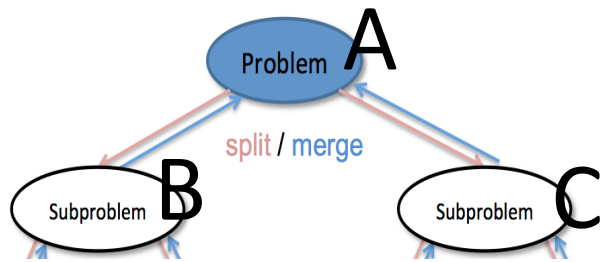
Quick sort

1. Из массива A делаем два подмассива B и C
   - Выбираем *pivot* – опорный элемент
   - Выполняем *partition*:
     - (B, C) = partition(A, pivot)



Quick sort

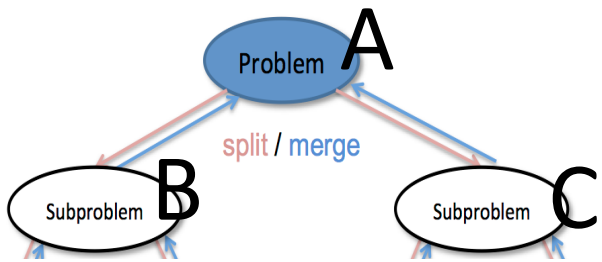
1. Из массива A делаем два подмассива B и C
   - Выбираем *pivot* – опорный элемент
   - Выполняем *partition*:
     - (B, C) = partition(A, pivot)



B – ключи «≤ pivot»
C – ключи «≥ pivot»

Quick sort

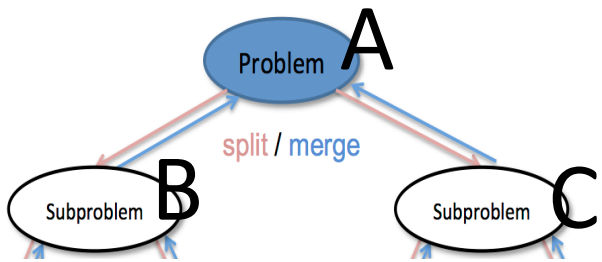
1. Из массива A делаем два подмассива B и C
   - Выбираем *pivot* – опорный элемент
   - Выполняем *partition*:
     - (B, C) = partition(A, pivot)
2. Решаем задачу рекурсивно B -> S(B), C -> S(C)



B – ключи «≤ pivot»
C – ключи «≥ pivot»

Quick sort

1. Из массива A делаем два подмассива B и C
   - Выбираем *pivot* – опорный элемент
   - Выполняем *partition*:
     - (B, C) = partition(A, pivot)
2. Решаем задачу рекурсивно B -> S(B), C -> S(C)
3. Конкатенируем отсортированные подмассивы
   - S(A) = S(B) + S(C)



B – ключи «≤ pivot»
C – ключи «≥ pivot»

Quick sort: детали

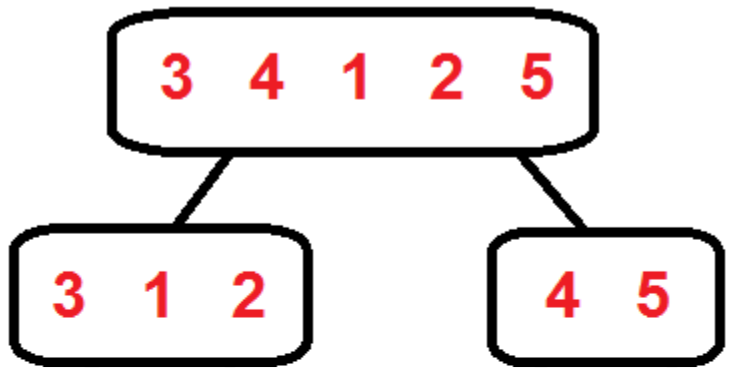
- Как выбрать pivot?
- Как реализовать partition?

Quick sort: выбор опорного элемента

- Рассмотрим дерево рекурсии данного алгоритма

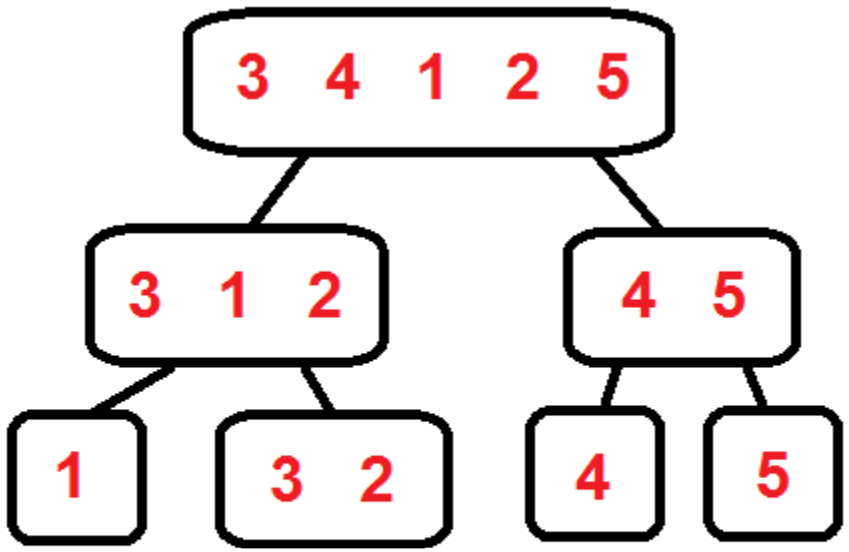
Quick sort: выбор опорного элемента

- Рассмотрим дерево рекурсии данного алгоритма



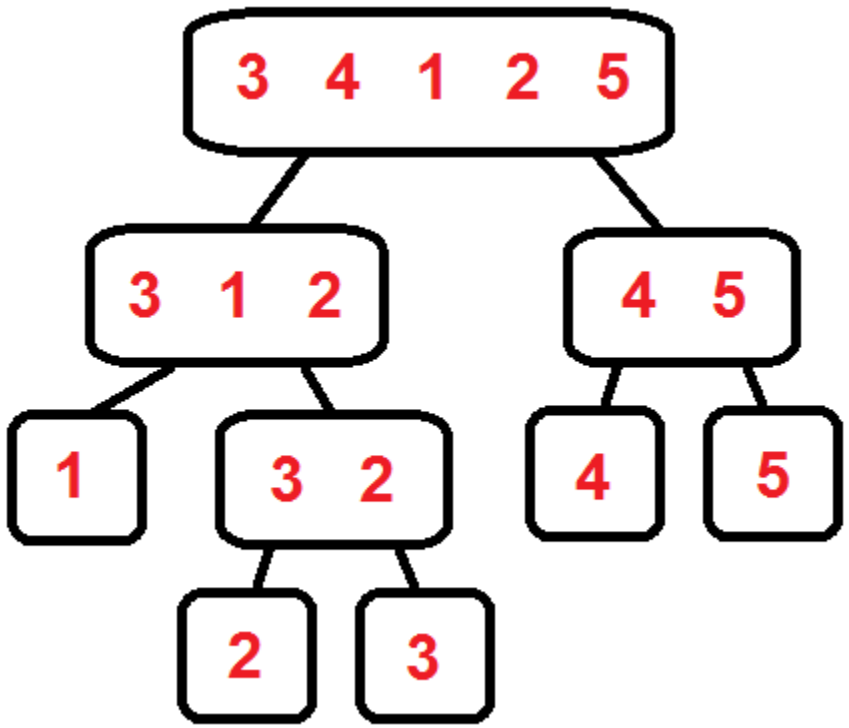
Quick sort: выбор опорного элемента

- Рассмотрим дерево рекурсии данного алгоритма



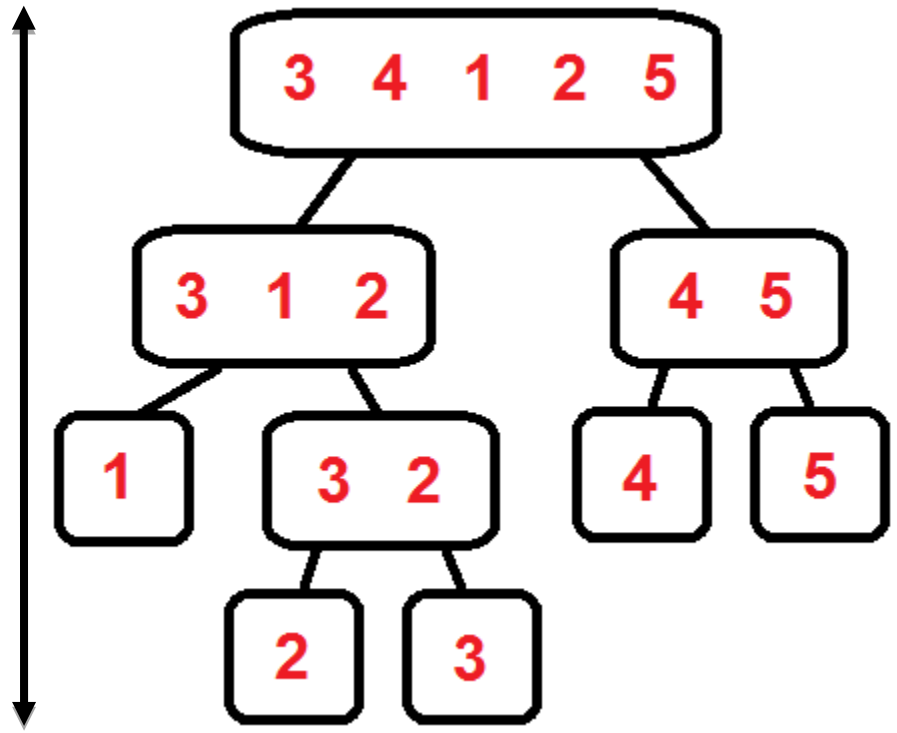
Quick sort: выбор опорного элемента

- Рассмотрим дерево рекурсии данного алгоритма



Quick sort: выбор опорного элемента

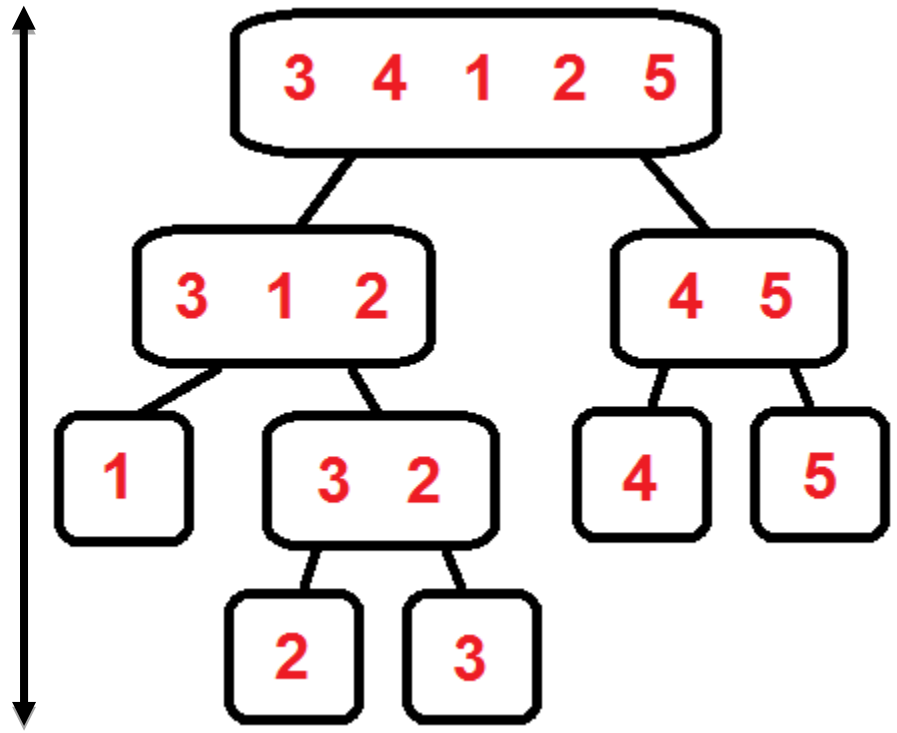
- Рассмотрим дерево рекурсии данного алгоритма
  - Высота дерева H
  - Число элементов на каждом «этаже» не превосходит N



Quick sort: выбор опорного элемента

- Рассмотрим дерево рекурсии данного алгоритма
  - Высота дерева H
  - Число элементов на каждом «этаже» не превосходит N

T = O(H * N)

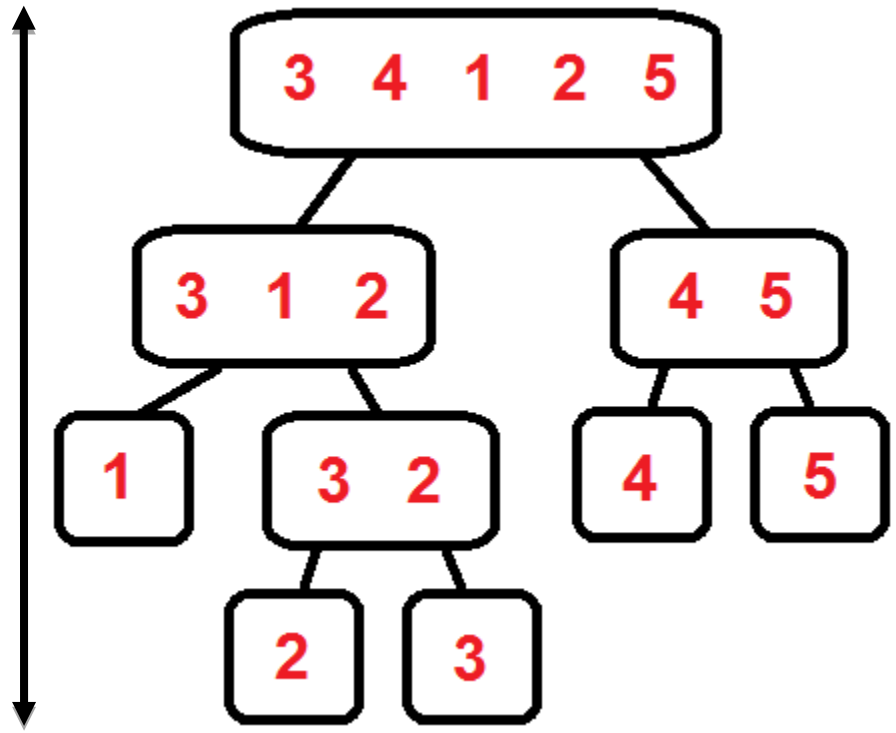


Quick sort: выбор опорного элемента

- Рассмотрим дерево рекурсии данного алгоритма
  - Высота дерева H
  - Число элементов на каждом «этаже» не превосходит N

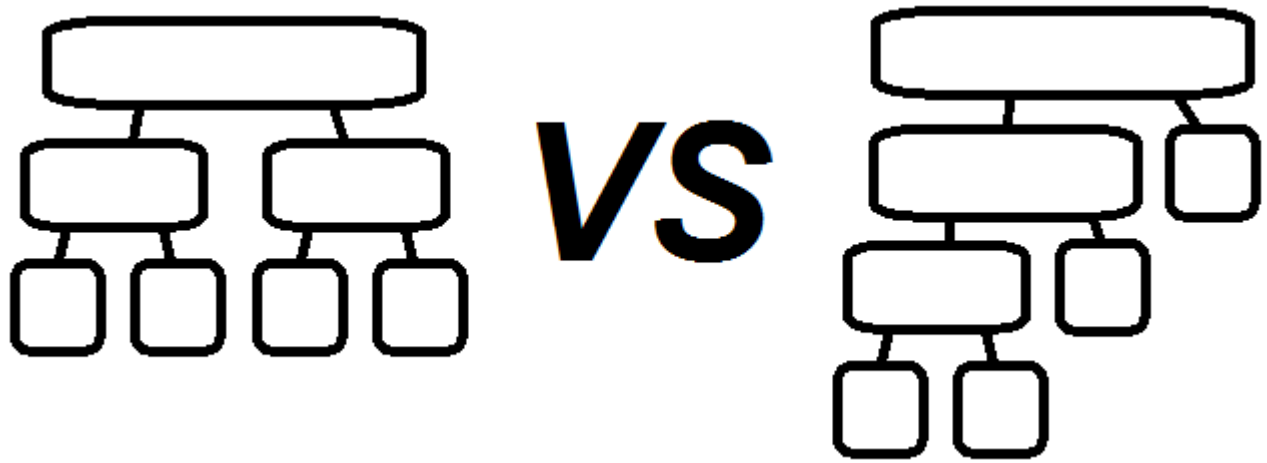
T = O(H * N)

- Структуру дерева определяет выбор опорного элемента



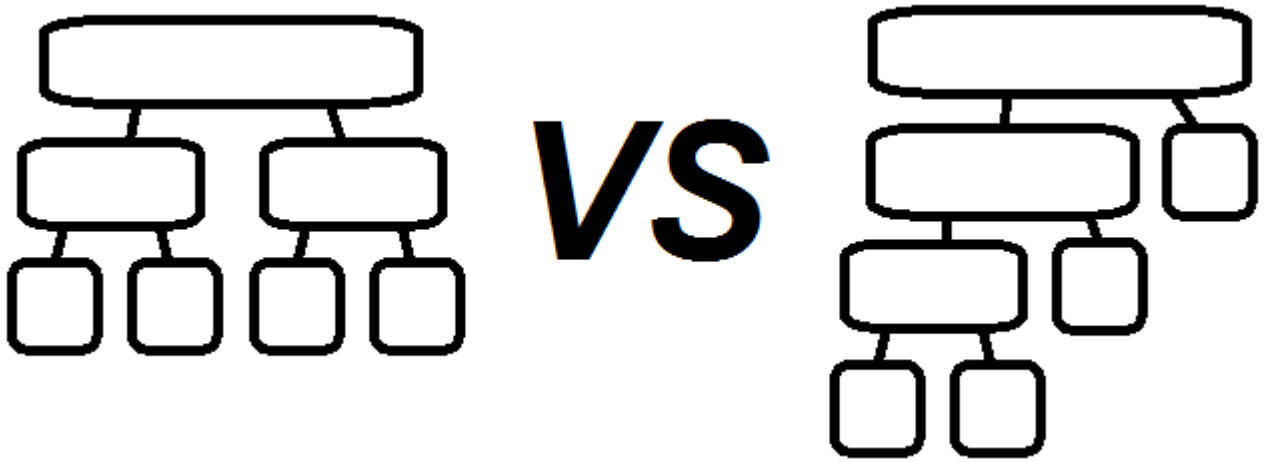
Quick sort: выбор опорного элемента

Структура оптимального дерева



Quick sort: выбор опорного элемента

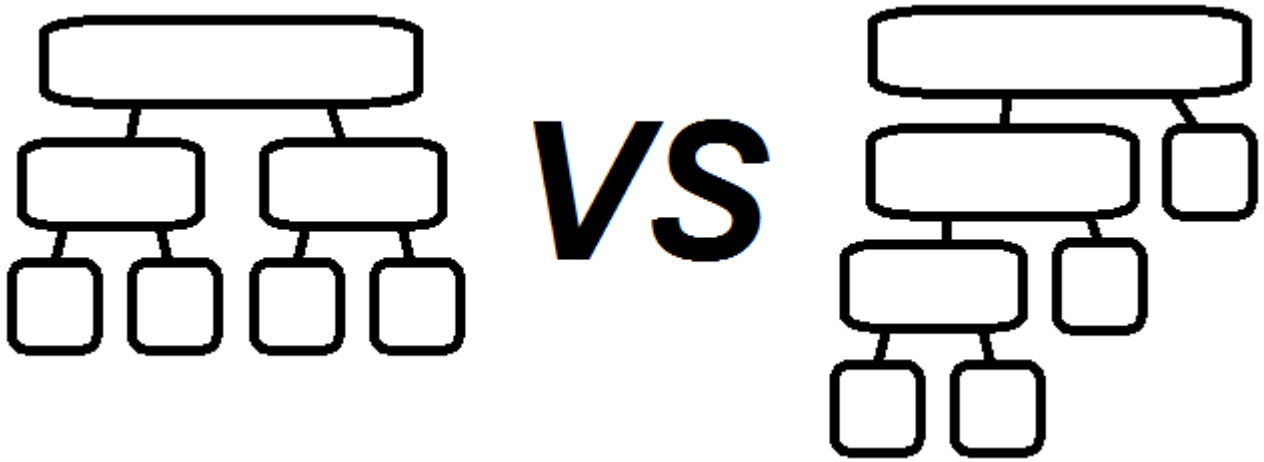
Структура оптимального дерева



pivot = медиана => идеальная структура

Quick sort: выбор опорного элемента

Структура оптимального дерева



pivot = медиана => идеальная структура

- H = log(N), T = O(N logN)

Quick sort: выбор опорного элемента

pivot = медиана => идеальная структура

Quick sort: выбор опорного элемента

pivot = медиана => идеальная структура

Проблама: искать медиану слишком затратно

Quick sort: выбор опорного элемента

pivot = медиана => идеальная структура

Проблама: искать медиану слишком затратно

Решение: рандомизация

Quick sort: выбор опорного элемента

pivot = медиана => идеальная структура

Проблама: искать медиану слишком затратно

Решение: рандомизация

- Выбирает pivot случайно среди всех элементов (равновероятно)

Quick sort: выбор опорного элемента

pivot = медиана => идеальная структура

Проблама: искать медиану слишком затратно

Решение: рандомизация

- Выбирает pivot случайно среди всех элементов (равновероятно)

Какое время работы рандомизированного алгоритма?

Quick sort: выбор опорного элемента

pivot = медиана => идеальная структура

Проблама: искать медиану слишком затратно

Решение: рандомизация

- Выбирает pivot случайно среди всех элементов (равновероятно)

Какое время работы рандомизированного алгоритма?

- Упражнение: оценить матожидание $E_K[\ T(N, K)\ ]$

  T(N, K) = время работы алгоритма, если разбиваем на K и (N - K)

Quick sort: partition

(B, C) = partition(A, pivot): B – ключи «≤ pivot», C – ключи «≥ pivot»

Quick sort: partition

(B, C) = partition(A, pivot): B – ключи «≤ pivot», C – ключи «≥ pivot»

1. In-place реализация

Quick sort: partition

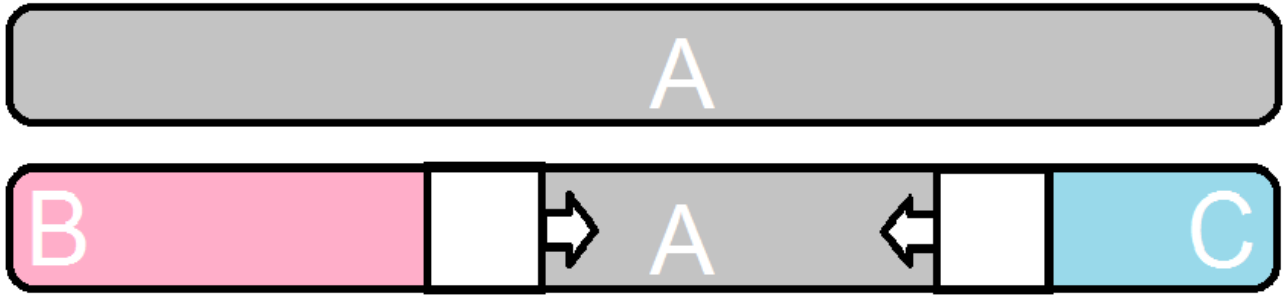
(B, C) = partition(A, pivot): B – ключи «≤ pivot», C – ключи «≥ pivot»

1. In-place реализация

Quick sort: partition

(B, C) = partition(A, pivot): B – ключи «≤ pivot», C – ключи «≥ pivot»

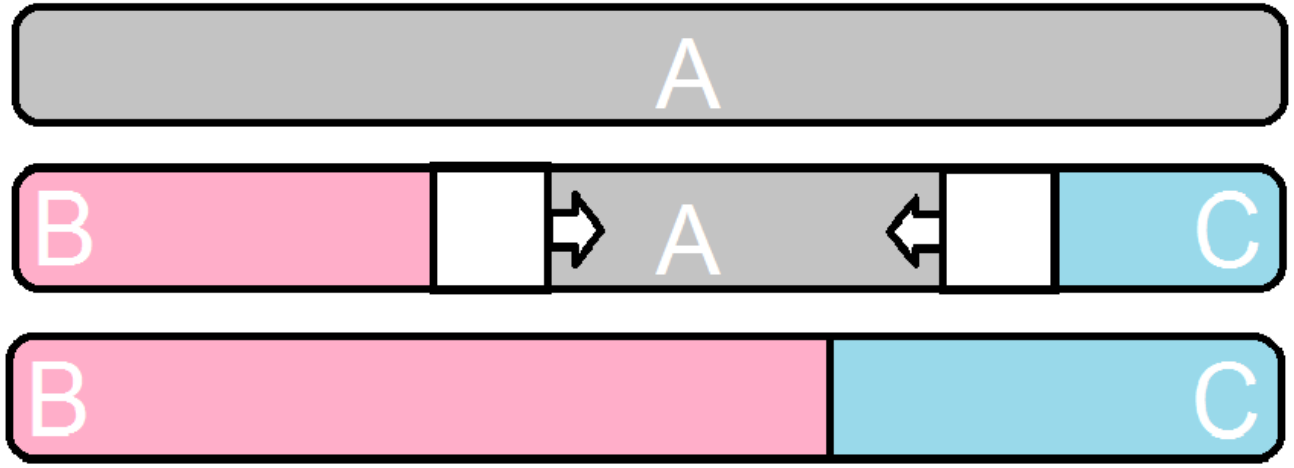
1. In-place реализация



Quick sort: partition

(B, C) = partition(A, pivot): B – ключи «≤ pivot», C – ключи «≥ pivot»

1. In-place реализация



Quick sort: partition

(B, C) = partition(A, pivot): B – ключи «≤ pivot», C – ключи «≥ pivot»
2. Lomuto partition

Quick sort: partition

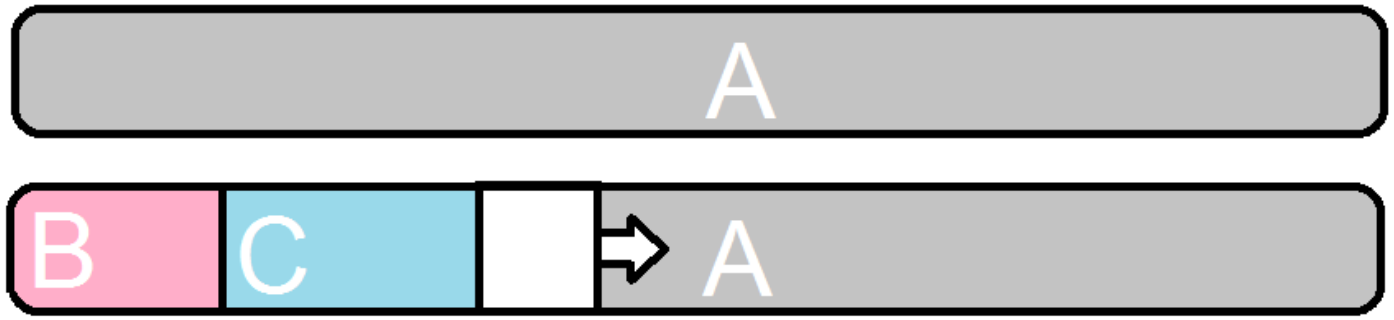
(B, C) = partition(A, pivot): B – ключи «≤ pivot», C – ключи «≥ pivot»

2. Lomuto partition

Quick sort: partition

(B, C) = partition(A, pivot): B – ключи «≤ pivot», C – ключи «≥ pivot»

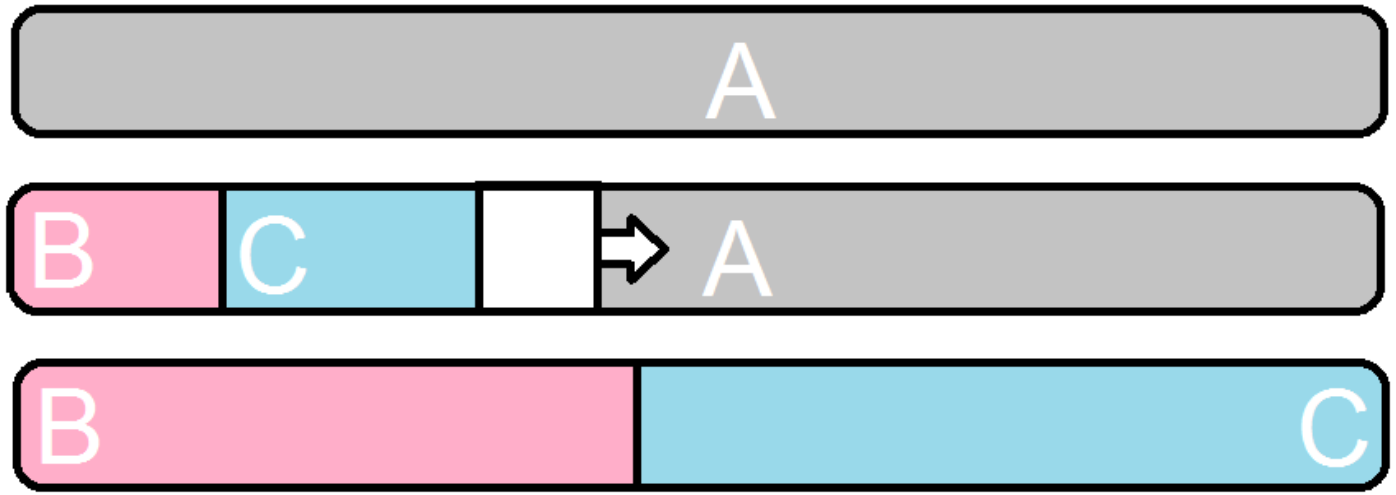
2. Lomuto partition



Quick sort: partition

(B, C) = partition(A, pivot): B – ключи «≤ pivot», C – ключи «≥ pivot»

2. Lomuto partition

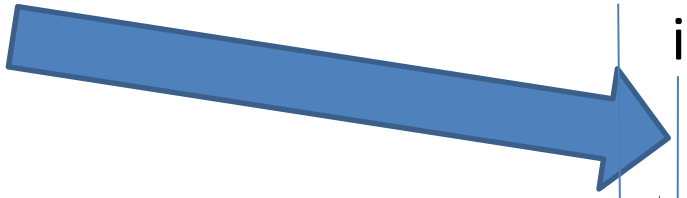


Quick sort: реализация

```
S(A) {
    if (|A| == 1) {
        return
    }
    pivot = pick_pivot(A)
    (B, C) = partition(A, pivot)
    S(B)
    S(C)
}
```

Quick sort: реализация

База рекурсии: простой алгоритм сортировки



```
S(A) {
    if (|A| ≤ n) {
        S_simple(A)
        return
    }
    pivot = pick_pivot(A)
    (B, C) = partition(A, pivot)
    S(B)
    S(C)
}
```

Quick sort: реализация

База рекурсии: простой алгоритм сортировки

```
S(A) {
    if (|A| ≤ n) {
        S_simple(A)
        return
    }
    pivot = pick_pivot(A)
    (B, C) = partition(A, pivot)
    S(B)
    S(C)
}
```

Quick sort: реализация

База рекурсии: простой алгоритм сортировки
Проблема с глубиной рекурсии

```
S(A) {
    if (|A| ≤ n) {
        S_simple(A)
        return
    }
    pivot = pick_pivot(A)
    (B, C) = partition(A, pivot)
    S(B)
    S(C)
}
```

Quick sort: реализация

База рекурсии: простой алгоритм сортировки
Проблема с глубиной рекурсии
Решение: удаление хвостовой рекурсии

```
S(A) {
    if (|A| ≤ n) {
        S_simple(A)
        return
    }
    pivot = pick_pivot(A)
    (B, C) = partition(A, pivot)
    S(B)
    S(C)
}
```

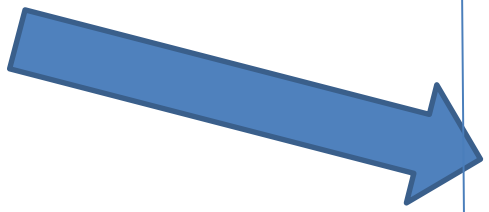
Quick sort: реализация

База рекурсии: простой алгоритм сортировки

Проблема с глубиной рекурсии

Решение: удаление хвостовой рекурсии

- S(A) в конце вызывает S(B) и S(C)



```
S(A) {
    if (|A| ≤ n) {
        S_simple(A)
        return
    }
    pivot = pick_pivot(A)
    (B, C) = partition(A, pivot)
    S(B)
    S(C)
}
```

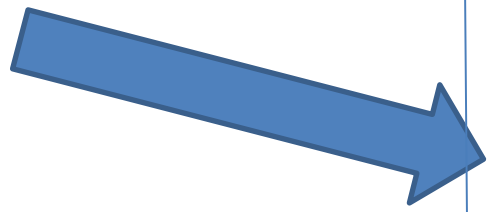
Quick sort: реализация

База рекурсии: простой алгоритм сортировки

Проблема с глубиной рекурсии

Решение: удаление хвостовой рекурсии

- S(A) в конце вызывает S(B) и S(C)
- Пусть |B| ≤ |C|



```
S(A) {
    if (|A| ≤ n) {
        S_simple(A)
        return
    }
    pivot = pick_pivot(A)
    (B, C) = partition(A, pivot)
    S(B)
    S(C)
}
```

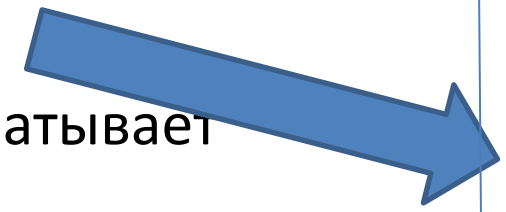
Quick sort: реализация

База рекурсии: простой алгоритм сортировки

Проблема с глубиной рекурсии

Решение: удаление хвостовой рекурсии

- S(A) в конце вызывает S(B) и S(C)
- Пусть |B| ≤ |C|
  - Тогда S(A) вызывает S(B), а C обрабатывает в цикле



```
S(A) {
    if (|A| ≤ n) {
        S_simple(A)
        return
    }
    pivot = pick_pivot(A)
    (B, C) = partition(A, pivot)
    S(B)
    S(C)
}
```

Quick sort: реализация

Пусть |B| ≤ |C|

- Тогда S(A) вызывает S(B), а C обрабатывает в цикле

```
S(A) {
    while (|A| > n) {
        pivot = pick_pivot(A)
        (B, C) = partition(A, pivot)
        if (|B| ≤ |C|)
            (A, D) = (C, B)
        else
            (A, D) = (B, C)
        S(D)
    }
    S_simple(A)
}
```

Quick sort: реализация

Пусть |B| ≤ |C|

- Тогда S(A) вызывает S(B), а C обрабатывает в цикле

Заменили рекурсию итерацией

```
S(A) {
    while (|A| > n) {
        pivot = pick_pivot(A)
        (B, C) = partition(A, pivot)
        if (|B| ≤ |C|)
            (A, D) = (C, B)
        else
            (A, D) = (B, C)
        S(D)
    }
    S_simple(A)
}
```

Quick sort: реализация

Пусть |B| ≤ |C|

- Тогда S(A) вызывает S(B), а C обрабатывает в цикле

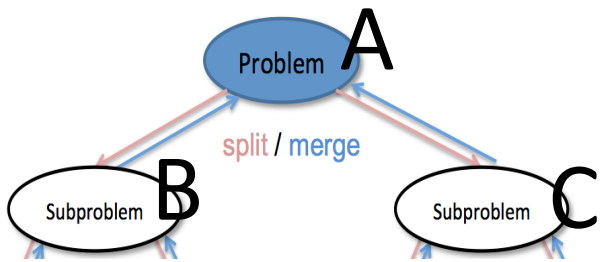
Заменили рекурсию итерацией

Теперь глубина рекурсии ≤ logN

```
S(A) {
    while (|A| > n) {
        pivot = pick_pivot(A)
        (B, C) = partition(A, pivot)
        if (|B| ≤ |C|)
            (A, D) = (C, B)
        else
            (A, D) = (B, C)
        S(D)
    }
    S_simple(A)
}
```

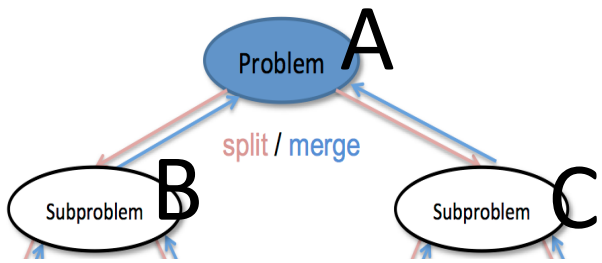
Merge sort

1. Из массива A делаем два подмассива B и C
   - B – первая половина массива, C – вторая



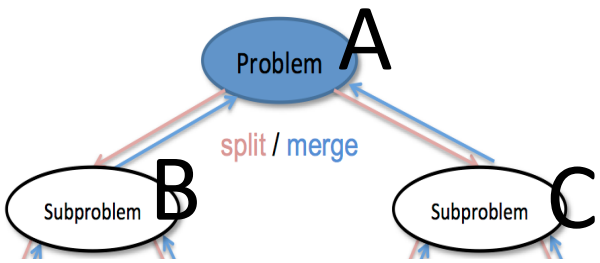
Merge sort

1. Из массива A делаем два подмассива B и C
   - B – первая половина массива, C – вторая
2. Решаем задачу рекурсивно B -> S(B), C -> S(C)

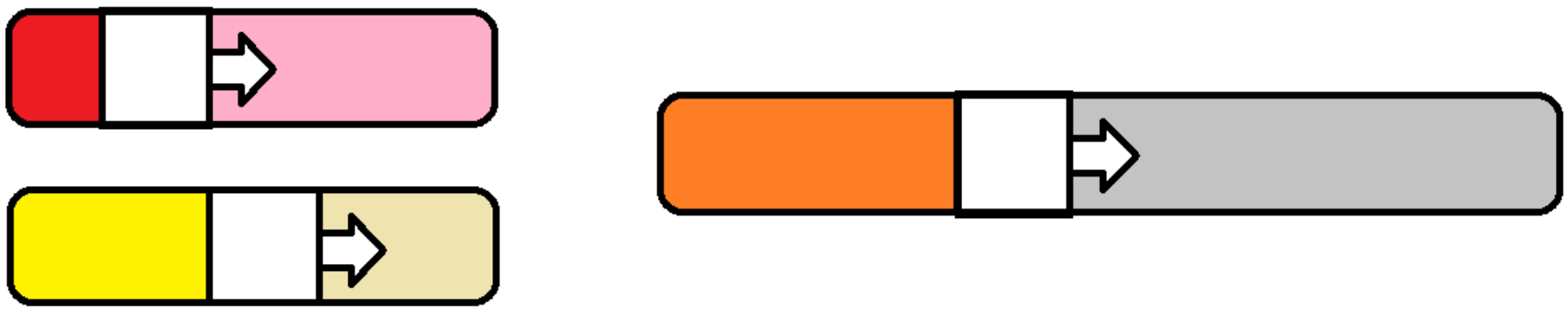


Merge sort

1. Из массива A делаем два подмассива B и C
   - B – первая половина массива, C – вторая
2. Решаем задачу рекурсивно B -> S(B), C -> S(C)
3. Сливаем отсортированные подмассивы
   - S(A) = merge(S(B), S(C))



Merge sort: merge



Merge sort: время работы

Глубина дерева рекурсии = log(N)

Merge sort: время работы

Глубина дерева рекурсии = log(N)
T = O(N log(N))

Merge sort: свойства

In-place?
Stable?

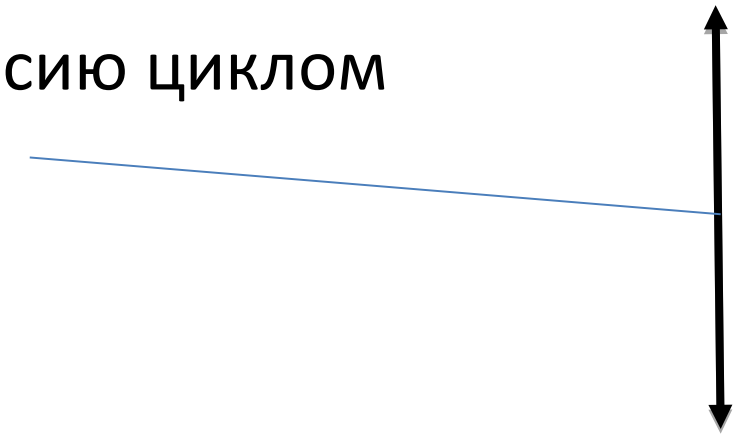
Merge sort: избавление от рекурсии

Заменяем рекурсию циклом

Merge sort: избавление от рекурсии

Заменяем рекурсию циклом

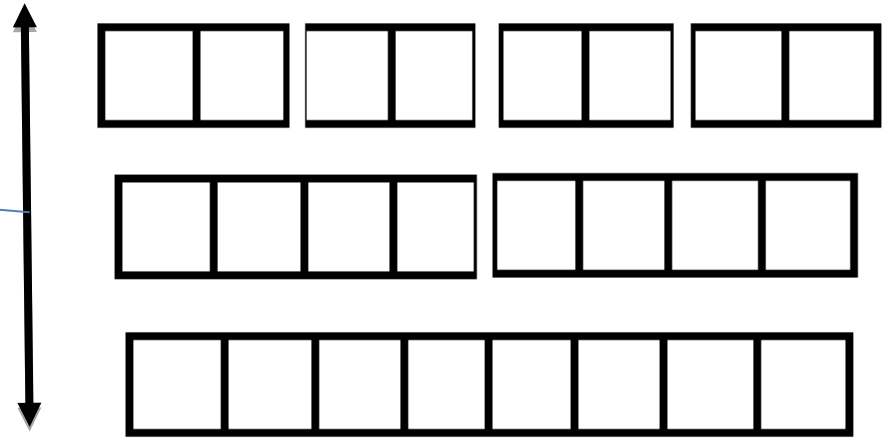
- logN итераций



Merge sort: избавление от рекурсии

Заменяем рекурсию циклом

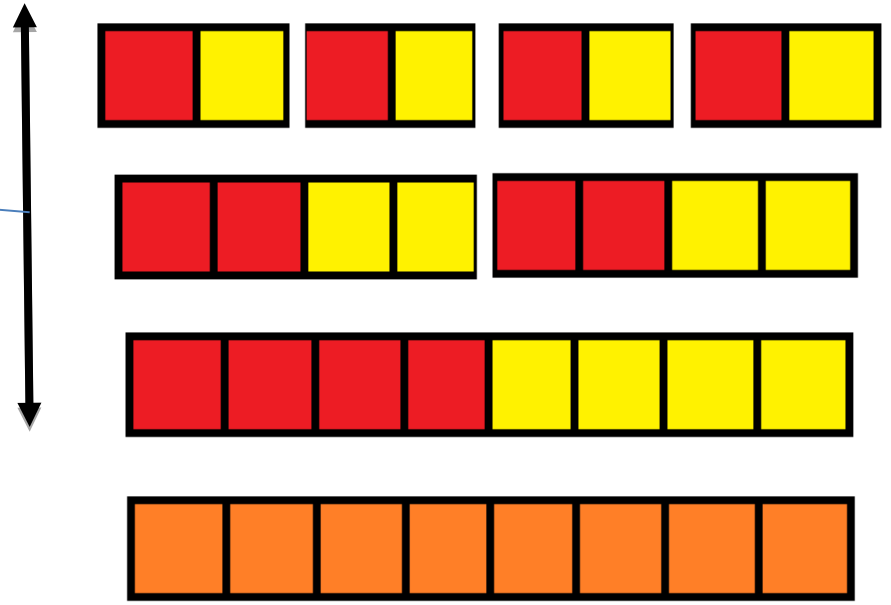
- logN итераций
- На K-ой итерации
  - Разбиваем массив на блоки |b|=2^K



Merge sort: избавление от рекурсии

Заменяем рекурсию циклом

- logN итераций
- На K-ой итерации
  - Разбиваем массив на блоки |b|=2^K
  - Сливаем пары соседних блоков



Merge sort: in-place

Попробуем использовать O(1) дополнительной памяти

Merge sort: in-place

Попробуем использовать O(1) дополнительной памяти
Что если использовать в качестве промежуточного буфера фрагмент нашего массива?

Merge sort: in-place

Попробуем использовать O(1) дополнительной памяти
Что если использовать в качестве промежуточного буфера фрагмент нашего массива?

- Нельзя присваивать, затирая старое значение

Merge sort: in-place

Попробуем использовать O(1) дополнительной памяти

Что если использовать в качестве промежуточного буфера фрагмент нашего массива?

- Нельзя присваивать, затирая старое значение
- Но можно совершать обмен (swap)

Merge sort: in-place

- Можно отсортировать половину массива

Merge sort: in-place

- Можно отсортировать половину массива
- И ещё четверть

Merge sort: in-place

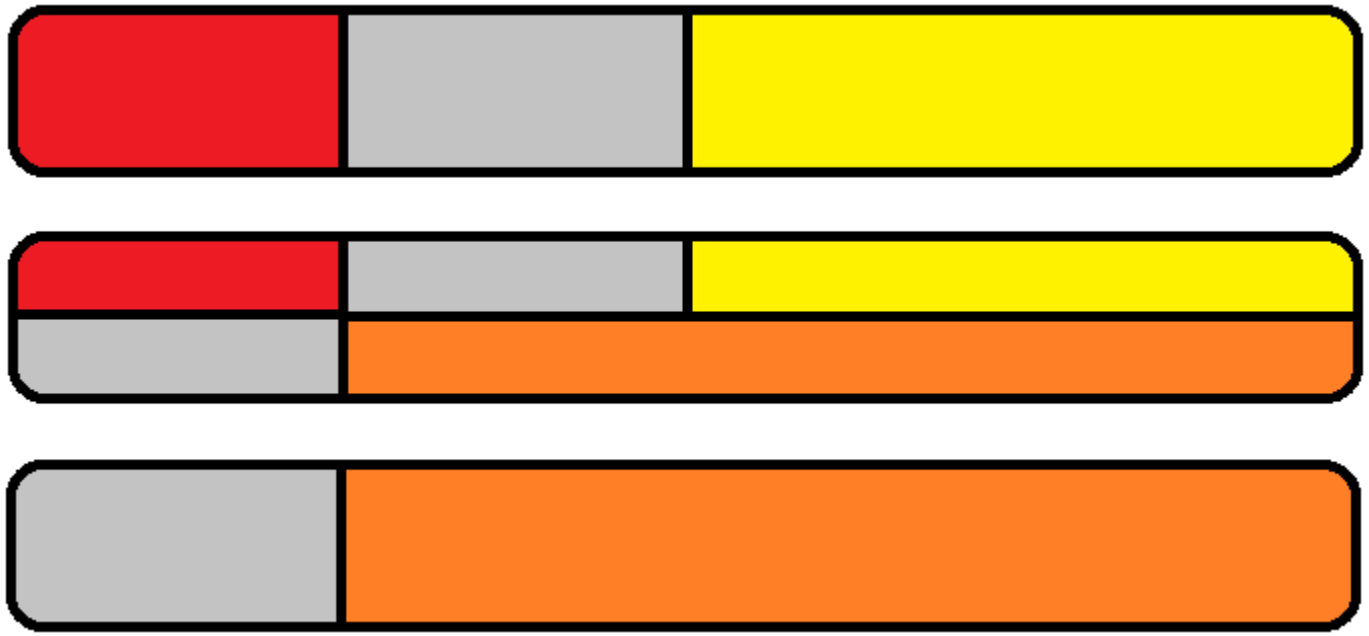
Как слить половину и четверть?

Merge sort: in-place

Как слить половину и четверть?

Merge sort: in-place

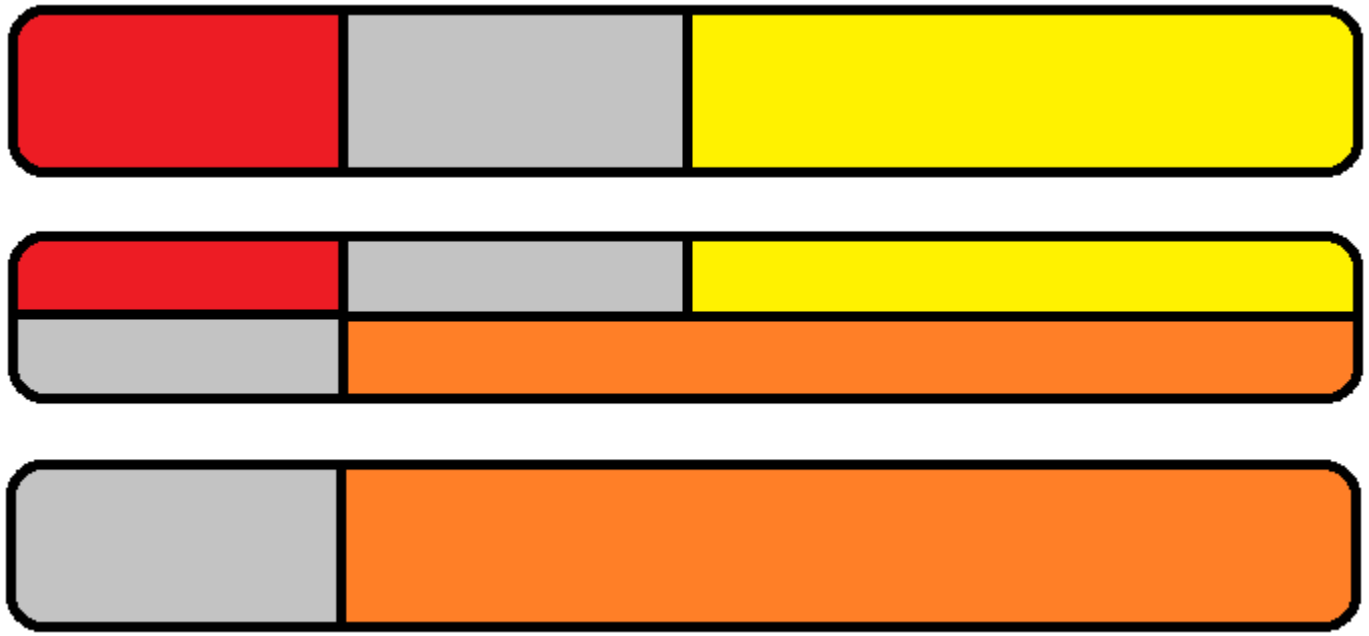
Как слить половину и четверть?



Merge sort: in-place

Как слить половину и четверть?

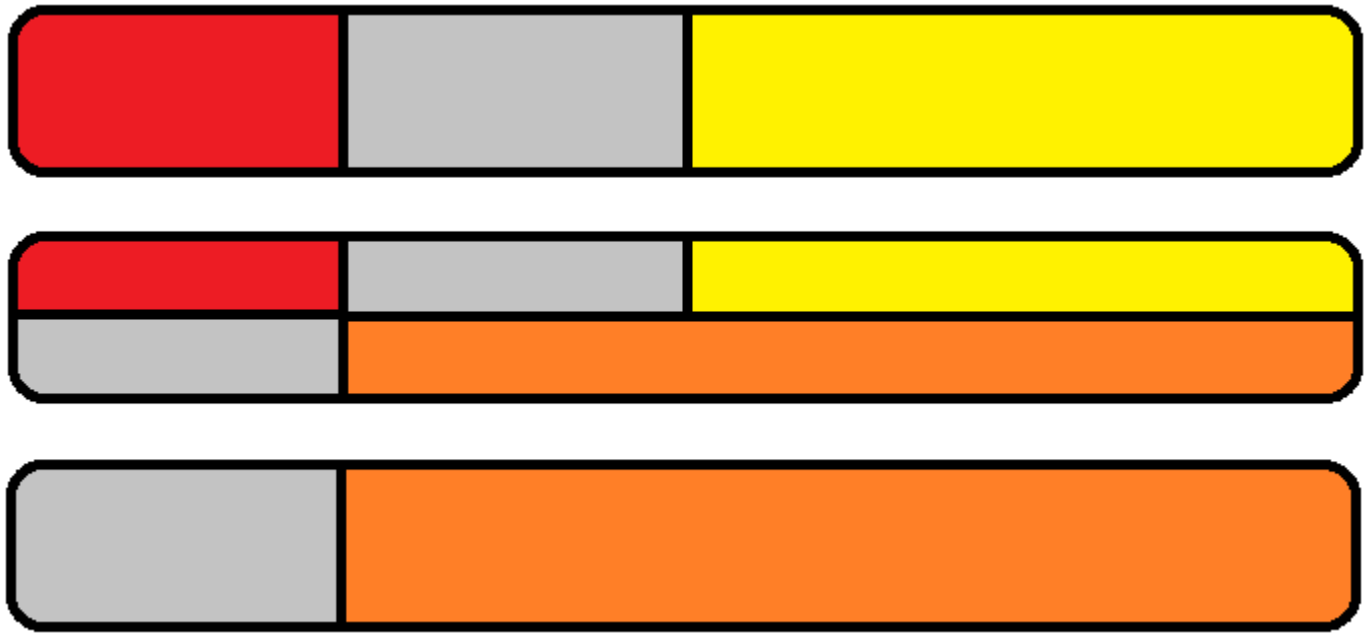
Время на слияние = ¾N



Merge sort: in-place

Как слить половину и четверть?
Время на слияние = ¾N

- Продолжаем для (7/8)N, и.т.д.

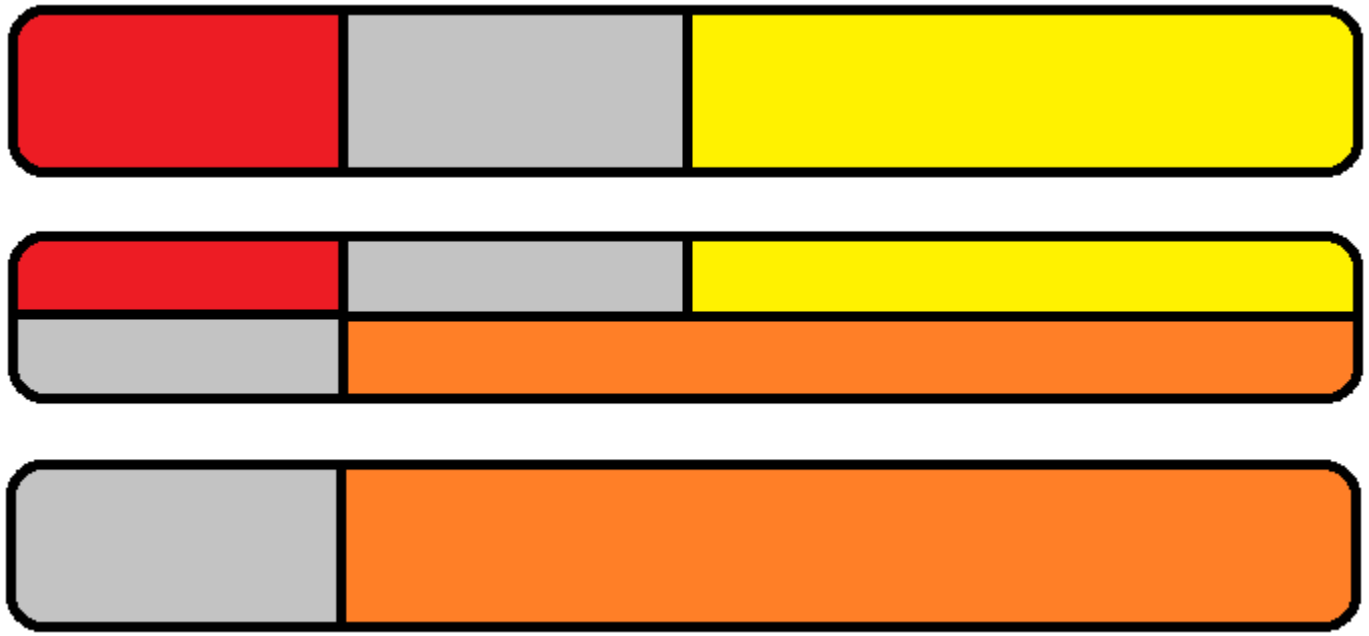


Merge sort: in-place

Как слить половину и четверть?
Время на слияние = ¾N

- Продолжаем для (7/8)N, и.т.д.

T = O(NlogN)



Оптимальное дерево слияний

Собираемся слить M отсортированных последовательностей

Оптимальное дерево слияний

Собираемся слить M отсортированных последовательностей

- их длины N_1, … N_M

Оптимальное дерево слияний

Собираемся слить M отсортированных последовательностей

- их длины N_1, … N_M

Время на слияние с длинами N_i и N_j составляет $O(N_i + N_j)$

Оптимальное дерево слияний

Собираемся слить M отсортированных последовательностей

- их длины N_1, … N_M

Время на слияние с длинами N_i и N_j составляет $O(N_i + N_j)$

- В каком порядке их оптимально сливать?

Оптимальное дерево слияний

Собираемся слить M отсортированных последовательностей

- их длины N_1, … N_M

Время на слияние с длинами N_i и N_j составляет $O(N_i + N_j)$

- В каком порядке их оптимально сливать?

Идея:

- сольём две наиболее которких

Оптимальное дерево слияний

Собираемся слить M отсортированных последовательностей

- их длины N_1, … N_M

Время на слияние с длинами N_i и N_j составляет $O(N_i + N_j)$

- В каком порядке их оптимально сливать?

Идея:

- сольём две наиболее которких
- теперь у нас M – 1 последовательность

Оптимальное дерево слияний

Собираемся слить M отсортированных последовательностей

- их длины N_1, … N_M

Время на слияние с длинами N_i и N_j составляет $O(N_i + N_j)$

- В каком порядке их оптимально сливать?

Идея:

- сольём две наиболее которких
- теперь у нас M – 1 последовательность
- продолжим действовать пока не сольём все

Оптимальное дерево слияний и коды Хаффмана

Слить M последовательностей с длинами N_1, … N_M:

- на каждом шаге сливаем две наиболее которких, заменяя из на новую

Как назначить префиксные коды M символам с частотами P_1, … P_M, чтобы сократить длину закодированной полседовательности?

Оптимальное дерево слияний и коды Хаффмана

Слить M последовательностей с длинами N_1, … N_M:

- на каждом шаге сливаем две наиболее которких, заменяя из на новую

Как назначить префиксные коды M символам с частотами P_1, … P_M, чтобы сократить длину закодированной полседовательности?

- Код префиксный если для любых символов код одного не префикс другого

Оптимальное дерево слияний и коды Хаффмана

Слить M последовательностей с длинами N_1, … N_M:

- на каждом шаге сливаем две наиболее которких, заменяя из на новую

Как назначить префиксные коды M символам с частотами P_1, … P_M, чтобы сократить длину закодированной полседовательности?

- Код префиксный если для любых символов код одного не префикс другого
- Стремимся уменьшить $\sum P_i L_i$, L_i – длина кода i-ого символа

Оптимальное дерево слияний и коды Хаффмана

Как назначить префиксные коды M символам с частотами P_1, … P_M, чтобы сократить длину закодированной полседовательности?

- Код префиксный если для любых символов код одного не префикс другого
- Стремимся уменьшить $\sum P_i L_i$, L_i – длина кода i-ого символа

Оптимальное дерево слияний и коды Хаффмана

Как назначить префиксные коды M символам с частотами P_1, … P_M, чтобы сократить длину закодированной полседовательности?

- Код префиксный если для любых символов код одного не префикс другого
- Стремимся уменьшить $\sum P_i L_i$, L_i – длина кода i-ого символа

Идея:

- выберем два наиболее редких символа

Оптимальное дерево слияний и коды Хаффмана

Как назначить префиксные коды M символам с частотами P_1, … P_M, чтобы сократить длину закодированной полседовательности?

- Код префиксный если для любых символов код одного не префикс другого
- Стремимся уменьшить $\sum P_i L_i$, L_i – длина кода i-ого символа

Идея:

- выберем два наиболее редких символа
- увеличим длину их кода на единицу

Оптимальное дерево слияний и коды Хаффмана

Как назначить префиксные коды M символам с частотами P_1, … P_M, чтобы сократить длину закодированной полседовательности?

- Код префиксный если для любых символов код одного не префикс другого
- Стремимся уменьшить $\sum P_i L_i$, L_i – длина кода i-ого символа

Идея:

- выберем два наиболее редких символа
- увеличим длину их кода на единицу
- заменим их на новый символ, теперь M – 1 символов

Оптимальное дерево слияний и коды Хаффмана

Как назначить префиксные коды M символам с частотами P_1, … P_M, чтобы сократить длину закодированной полседовательности?

- Код префиксный если для любых символов код одного не префикс другого
- Стремимся уменьшить $\sum P_i L_i$, L_i – длина кода i-ого символа

Идея:

- выберем два наиболее редких символа
- увеличим длину их кода на единицу
- заменим их на новый символ, теперь M – 1 символов
- продолжим действовать пока не останется один символ

Оптимальное дерево слияний и коды Хаффмана

Слить M последовательностей с длинами N_1, … N_M:

- на каждом шаге сливаем две наиболее которких, заменяя из на новую

Назначить префиксные коды M символам с частотами P_1, … P_M:

- на каждом шаге объединяем два наиболее редких символа

Оптимальное дерево слияний и коды Хаффмана

Слить M последовательностей с длинами N_1, … N_M:

- на каждом шаге сливаем две наиболее которких, заменяя из на новую

$T = \sum N_i H_i$

H_i – глубина в дереве слияний

Назначить префиксные коды M символам с частотами P_1, … P_M:

- на каждом шаге объединяем два наиболее редких символа

$L = \sum P_i L_i$

L_i – глубина в дереве кодирования