

# Семинар №9

ФАКИ 2017

---

Бирюков В. А.

November 8, 2017

# Управление памятью

## Сегменты памяти процесса

- Стек представляет собой обычный алгоритмический стек, применённый для управления памяти
- В нём хранятся локальные переменные
- Имеет фиксированный размер, определяется операционной системой, на порядок меньше чем Куча
- Немного быстрее, чем Куча

- Куча представляет собой обычную алгоритмическую кучу, применённую для управления памяти
- В ней можно динамически выделять память
- Размер, обычно, ограничен только доступными ресурсами
- Немного медленней, чем Стек

# Алгоритмы

---

- Алгоритм – это формально описанная вычислительная процедура, получающая исходные данные, и выдающая результат вычислений на выход (Кормен и др. "Алгоритмы: построение и анализ")

- Задана последовательность чисел
- Нужно найти такую перестановку исходной последовательности, чтобы элементы были расположены по возрастанию
- 5 2 4 6 1 3 2 9  $\rightarrow$  1 2 2 3 4 5 6 9

- Сортировка вставками
- Сортировка выбором
- Сортировка пузырьком



# Анализ алгоритмов

---

- Обычно изучают зависимость времени работы от размера входа
- Размер входа – зависит от конкретной задачи
- Для сортировки, размер входа – это количество элементов, которые нужно отсортировать
- Время работы – число элементарных шагов, которые выполняет алгоритм

```
for (int j = 0; j < length-1; j++)  
    for (int i = 0; i < length-1; i++)  
        if (a[i] > a[i+1])  
            swap(a[i], a[i+1]);
```

- Число операций, требуемых на один проход:  $a * n$
- Число проходов:  $n$
- Значит, время работы  $\sim n^2$

Принцип "разделяй и  
властвуй"

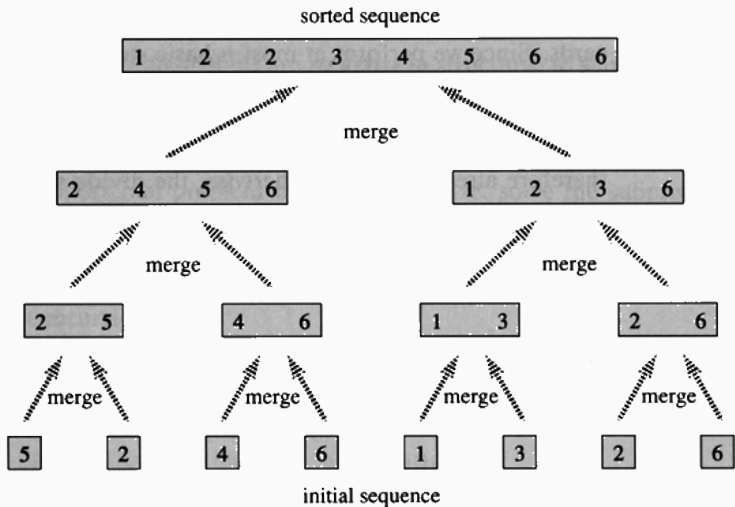
---

# Принцип "разделяй и властвуй"

- Задача разбивается на несколько подзадач меньшего размера
- Эти задачи решаются (обычно с помощью рекурсивного вызова)
- Решения этих задач комбинируются и получается решение исходной задачи

- Разбиваем массив на 2 половины
- Сортируем каждую половину
- Соединяем 2 упорядоченных массива в один

# Сортировка слиянием



```
void MergeSort(int * A, int p, int r)
{
    if (p < r)
    {
        int q = (p + r) / 2;
        MergeSort(A, p, q);
        MergeSort(A, q + 1, r);
        Merge(A, p, q, r);
    }
}
```



# Быстрая сортировка

---

# Быстрая сортировка (quicksort)

- Выбираем в массиве некоторый элемент, который будем называть опорным
- Переставляем элементы массива таким образом, чтобы все элементы со значением меньшим или равным опорному элементу, оказались слева от него, а все элементы, превышающие по значению опорный — справа от него
- Рекурсивно сортируем подмассивы, лежащие слева и справа от опорного элемента

# Быстрая сортировка (quicksort)



Неотсортированный массив



Опорное значение = 7



$12 \geq 7 \geq 2$ ,  
меняем местами 12 и 2



$26 \geq 7 \geq 7$   
меняем 26 и 7

# Быстрая сортировка (quicksort)



...



# Время работы сортировок

---

- Время работы сортировки пузырьком, выбором и вставками  $\sim n^2$
- Время работы сортировки слиянием и быстрой сортировки в среднем  $\sim n\log(n)$

- Пусть мы хотим отсортировать массив из 1 млн. чисел
- Сортировка пузырьком написана аккуратно и требует  $2n^2$  операций и выполняется на суперкомпьютере(x100)
- Сортировка слиянием написана неэффективно и требует  $50n\log(n)$  операций и выполняется на пк(x1)
- Сортировка пузырьком выполнится за 5.5 часов
- Сортировка слиянием выполнится за 17 минут

# Стандартная сортировка

## qsort()

---



# Стандартная сортировка qsort()

```
#include <stdlib.h>
int values[] = { 88, 56, 100, 2, 25 };

int cmp(const void * a, const void * b)
{
    return ( *(int*)a - *(int*)b );
}

...
qsort(values, 5, sizeof(int), cmp);
```