Семинар #7: Сортировки. Домашнее задание.

Сложность алгоритмов

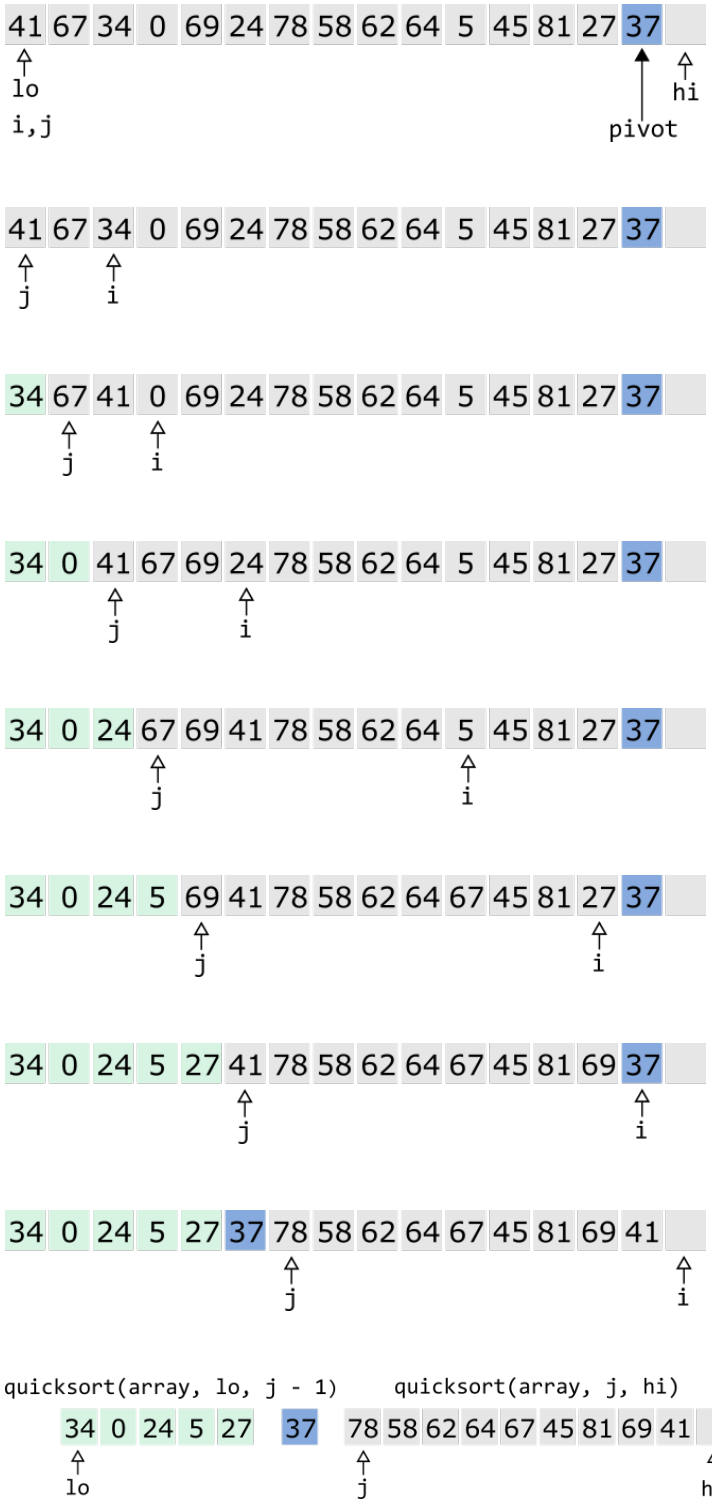
Чему равна средняя вычислительная сложность следующих операций?

1. Поиск элемента в неотсортированном массиве размера N
2. Поиск элемента в отсортированном массиве размера N (бинарный поиск)
3. Добавление элемента в начало массива размера N
4. Добавление элемента в конец массива размера N (в предположении, что `capacity` $>$ `size`)
5. Добавление элемента в динамический(саморасширяющийся) стек размера N при аддитивной стратегии выделения памяти (при нехватки места увеличиваем `capacity` на некоторую постоянную величину).
6. Добавление элемента в динамический стек размера N при мультипликативной стратегии выделения памяти (при нехватки места умножаем `capacity` на некоторую постоянную величину (обычно на 2)).
7. Сортировка выбором массива размера N
8. Сортировка пузырьком массива размера N
9. Быстрая сортировка массива размера N
10. Сортировка подсчётом массива размера N, если максимальный элемент массива равен K
11. Цифровая сортировка массива размера N, если максимальный элемент массива равен K
12. Сортировка Bogosort массива размера N
13. Сложение двух чисел длиной в N цифр (N может быть большим)
14. Простой алгоритм умножения(столбиком) двух чисел длиной в N цифр (N может быть большим)
15. Простой алгоритм проверки числа на простоту перебором от двух до корня этого числа. Число состоит из N цифр в десятичной записи (N может быть большим)
16.* Добавление элемента в двоичную кучу размера N
17.* Удаление элемента из двоичной кучи размера N

Нужно написать ответы в текстовый файл и прислать мне.

Быстрая сортировка - Quicksort

```c
#include <stdio.h>
#include <stdlib.h>
#define N 30

void quicksort(int* array, int lo, int hi)
{
    if (hi - lo > 1)
    {
        int j = lo;
        int pivot = array[hi - 1];
        for (int i = lo; i < hi; i++)
            if (array[i] <= pivot)
            {
                int temp = array[i];
                array[i] = array[j];
                array[j] = temp;
                j++;
            }

        quicksort(array, lo, j - 1);
        quicksort(array, j, hi);
    }
}

void print(int* array, int n)
{
    for (int i = 0; i < n; i++)
        printf("%d ", array[i]);
    printf("\n");
}

int main()
{
    int numbers[N];
    for(int i = 0; i < N; i++)
        numbers[i] = rand() % 100;

    print(numbers, N);
    quicksort(numbers, 0, N);
    print(numbers, N);
}
```



Задача: Звёзды: В файле `hipstars.csv` содержится информация о ближайших звёздах. Данные взяты из каталога Hipparcos. В каждой строке - информация об одной звезде:

1. `hip` - номер звезды в каталоге Hipparcos. Обратите внимание, что не все звёзды из каталога присутствуют в файле.
2. `proper_name` - традиционное имя звезды(строка не более чем 20 символов). Большинство звёзд имён не имеют и называются просто по номеру, например HIP 3345. Если у звезды имени нет, то в этом поле стоит прочерк `--`.
3. `right_ascension` и `declination` - прямое восхождение и склонение определяют положение звезды на небе. Аналог широты и долготы. (`double`)
4. `magnitude` - Звёздная величина - яркость звезды с точки зрения земного наблюдателя. Чем меньше, тем звезда ярче, шкала логарифмическая. Видимые глазом звёзды имеют звёздную величину 6 и ниже. Бетельгейзе $= 0.45$ Сириус $= -1.44$. Луна $= -12.7$. Солнце $= -26.7$. (`float`)
5. `absolute_magnitude` Абсолютная звёздная величина - яркость звезды с точки зрения наблюдателя, находящегося на растоянии в 10 парсек от этой звезды. Бетельгейзе $= -5.47$. Сириус $= 1.45$. Солнце $= 4.85$. (`float`)
6. `spectral_type` - спектральный класс звезды(2 символа).
7. `x`, `y` и `z` - Координаты звёзды в системе отсчёта, связанной с Землёй. Единица измерения - парсеки. 1 парсек $= 3.26$ световых года $= 206265$ расстояний от Земли до Солнца $= 3 \cdot 10^{16}$ метров. (`float`)
8. `constellation` - Созвездие (первые три буквы) или `NO`, если звезда не входит ни в какое созвездие.

- Опишите структуру `Star`, которая будет предназначена для хранения информации об одной звезде.
- **Считываем звёзды:**
  Формат `.csv` (comma-separated values) – это простейший формат для хранения табличных данных. В первой строке этого формата содержатся имена колонок таблицы, разделённые запятыми. Дальше – `n` строк таблицы, все величины в таблице разделены запятыми. Эти файлы можно открывать как в обычном текстовом редакторе, так и в программах для работы с таблицами (например, Excel). Если файл `hipstars.csv` открыть в текстовом редакторе, то его начало будет выглядеть примерно так:

```
hip,proper_name,ra,dec,mag,absmag,spectral_type,x,y,z,constellation
0,Sol,0.00000,0.00000,-26.70,4.85,G2,0.000005,0.000,0.000,NO
1,--,0.00090,1.08901,9.10,2.39,F5,219.741,0.003,4.177,Psc
2,--,0.00424,-19.49884,9.27,5.87,K3,45.211,0.003,-16.009,Cet
3,--,0.00503,38.85928,6.61,-1.62,B9,344.553,0.030,277.615,And
4,--,0.00853,-51.89355,8.06,2.42,F0,82.836,0.012,-105.620,Phe
```

  Для считывания используйте функцию `fscanf` из библиотеки `stdio.h`. Учтите, что спецификатор `%s` считывает строку до пробела. Чтобы считать строку до запятой используйте спецификатор `%[^,]` - при этом `s` на конец спецификатора ставить не надо. Пример считывания:

```c
#include <stdio.h>

int main()
{
    // Открываем файл hipstars.csv на чтение("r").
    FILE* f = fopen("hipstars.csv", "r");
    // Считываем первую строку
    char header[200];
    fscanf(f, "%s\n", header);
    // Считываем звезду
    fscanf(f, "%d,%[^,],%lf,%lf,%f,%f,%[^,],%f,%f,%f,%[^\n]\n", ...)
    // ...
    fclose(f);
}
```

Создайте массив из структур `Star` подходящего размера и считайте все данные из файла в массив. Файл содержит информацию о 114318 звезде, так что массив нужно создавать в куче (с помощью `malloc`).

- **Сохраняем звёзды:**
  Написать функцию `void save_stars(char* filename, char* header, Star* array, int n)`, которая будет сохранять звёзды из массива `array` в файл, чьё название хранится в переменной `filename`. Например, при вызове `save_stars(“output.csv”, header, stars, n);` массив `stars` должен сохраниться в файл `output.csv`. Первая строка в этом файле будет являться строкой `header`, в которой хранится название колонок.

- **Сортировка по видимой с Земли яркости:**
  Создайте функцию `quicksort_magnitude`, чтобы она принимала на вход массив из структур `Star` и сортировала их по возрастанию звёздной величины. Проверьте функцию в `main`, отсортировав структуры и сохранив их в файл `sorted_by_magnitude.csv` с помощью функции `save_stars`.

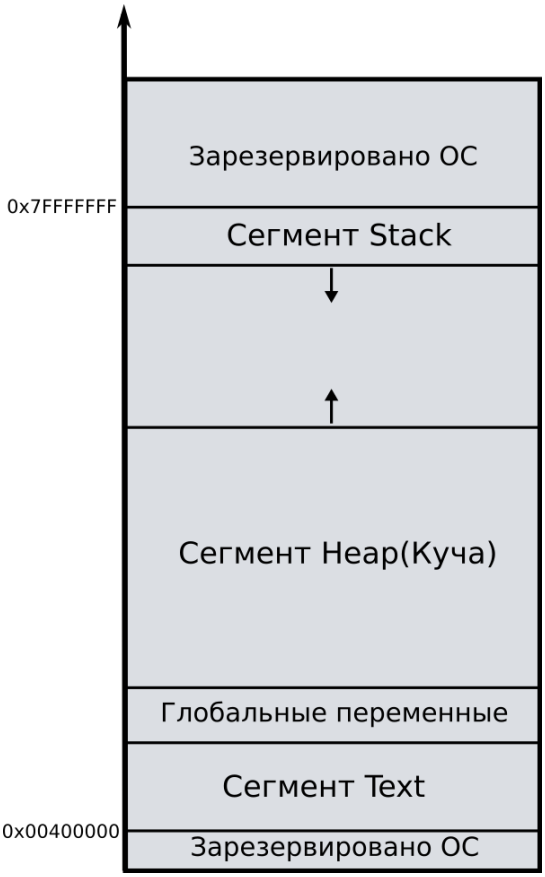
- **Сортировка по расстоянию:**
  Создайте функцию `quicksort_distance`, чтобы она сортировала массив звёзд по расстоянию от Земли. Проверьте функцию в `main`, отсортировав структуры и сохранив их в файл `sorted_by_distance.csv`.

- **Сортировка по температуре:**
  Создайте функцию `quicksort_temperature`, чтобы она сортировала массив звёзд по температуре поверхности. Темературу можно сравнить по первым двум символам спектра. Первый символ - спектральный класс звезды - от горячих к холодным: `O->B->A->F->G->K->M`. Второй символ - подкласс - число от 0 до 9, чем меньше, тем горячее. То есть теоретически самые горячие звёзды это звёзды класса `A0`, а самые холодные это звёзды класса `M9`. Проверьте функцию в `main`, отсортировав структуры и сохранив их в файл `sorted_by_temperature.csv`.
  Сложность этой задачи в том, что может быть много звёзд с одинаковым спектральным классом. Предположим, что подмассив для сортировки оказался тривиальным и все его элементы оказались равны друг другу. В этом случае наша `quicksort` не будет делить массив на 2 части (все элементы будут равны `pivot` и окажутся слева). В таком случае наша `quicksort` будет работать очень медленно. Чтобы этого избежать, вам нужно будет видоизменить функцию `quicksort` так, чтобы она делила массив на 3 части (элементы меньшие `pivot`, равные `pivot` и большие `pivot`) и продолжала сортировать только первую и третью части.

- **Функция-компаратор:**
  Объедините три предыдущие функции в одну с использованием функции-компаратора. Нужно написать функцию `void quicksort(Star* array, int lo, int hi, int (*cmp)(Star* a, Star* b))`, которая будет сортировать звёзды, основываясь на функции-компараторе `cmp`.
  Функция-компаратор принимает 2 указателя на элементы массива и возвращает:
  - положительное число, если `*b` должен идти в отсортированом массиве правее `*a`.
  - отрицательное число, если `*b` должен идти в отсортированом массиве левее `*a`.
  - `0`, если нужно сохранить текущий порядок элементов.

Эта задача очень похожа на задачу сортировки городов из классных задач. Решение можете посмотреть в папке `cities`. Там есть 2 программы: `sort_cities.c` – сортирует города и `sort_cities_funcpointer.c` – сортирует города с использованием функции компаратора.

Сегменты памяти. Указатели на функцию.



1. **Сегмент памяти Стек (Stack)**

   - При обычном объявлении переменных и массивов все они создаются в стеке: `int a;` или `int array[10];`
   - Память на эти переменные выделяется в начале функции и освобождается в конце функции.
   - Маленький размер (несколько мегабайт)
   - Выделение памяти происходит быстрее чем в куче

2. **Сегмент памяти Куча (Heap)**

   - `malloc` выделяет память в Куче.
     `int* p = (int*)malloc(10 * sizeof(int));`
   - Память выделяется при вызове `malloc` и освобождается при вызове `free`.
   - Размер ограничен свободной оперативной памятью - гигабайты.
   - Выделение памяти происходит медленней чем в стеке

3. **Сегмент памяти Text**

   - В этом сегменте хранится машинный код программы (Код на языке C, сначала, переводится в код на языке Ассемблера, а потом в машинный код. Как это происходит смотрите ниже.).
   - Адрес функции - адрес первого байта инструкций в этом сегменте.

Пример работы с указателем на функцию:

```c
#include <stdio.h>

void print(int a)
{
    printf("%d\n", a);
}
int main ()
{
    // Создадим указатель на функцию ( вместо названия функции - *p )
    void (*p)(int a) = print;

    // Теперь с p можно работать также как и с print
    p(123);
}
```

Подробней в файле `funcpointers/0funcpointer.c`.

Задачи на указатели на функцию:

- В файле `funcpointers/1foreach.c` лежит заготовка исходного кода. Вам нужно написать функцию `void foreach(int* array, int size, int (*f)(int))`, которая будет принимать на вход массив размера `size` и применять к каждому элементу функцию `f`.
- В файле `funcpointers/2foreach_second_argument.c` лежит заготовка исходного кода. Вам нужно написать функцию `void foreach(int* array, int size, int (*f)(int, int), int b)`, которая будет принимать на вход массив размера `size` и применять к каждому элементу функцию `g(x) = f(x, b)`.

Стандартная функция qsort

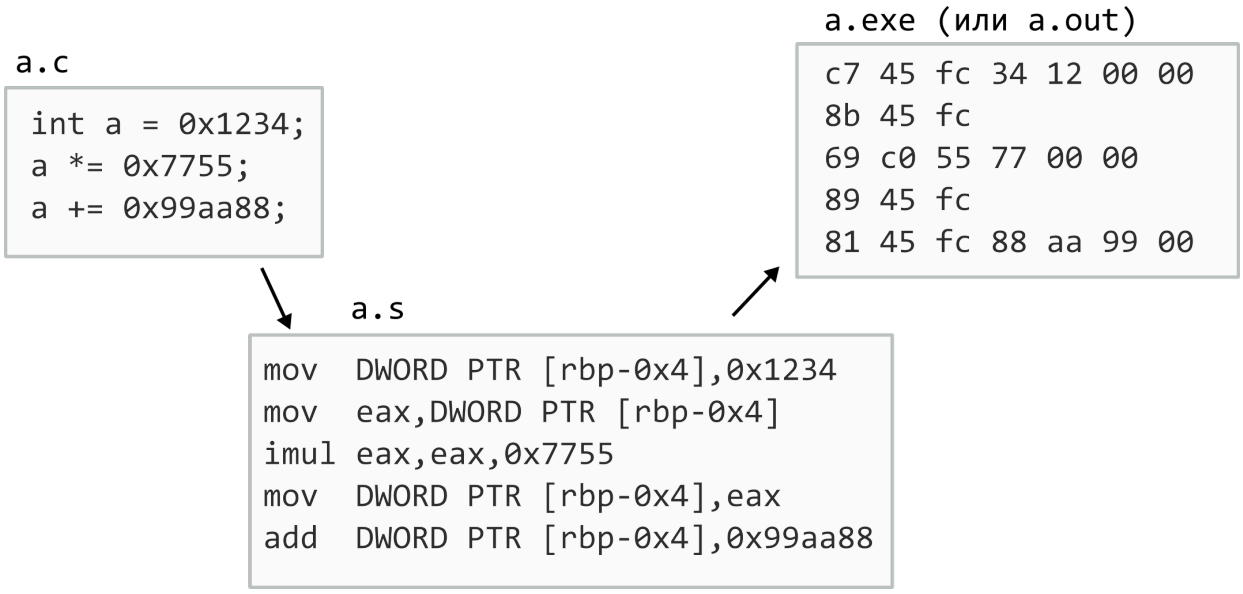
В библиотеке `stdlib.h` уже реализована функция `qsort`, которая сортирует произвольные элементы, используя быструю сортировку. Пример использования этой функции:

```
#include <stdio.h>
#include <stdlib.h>

int cmp(const void* a, const void* b)
{
    // В этот компаратор передаются указатели на void,
    // Поэтому их нужно привести в нужный нам тип:
    int* pa = (int*)a;
    int* pb = (int*)b;
    return (*pa - *pb);
}

int main()
{
    int arr[] = {163, 624, 7345, 545, 41, 78, 5, 536, 962, 1579};
    qsort(arr, 10, sizeof(int), cmp);
    // qsort( массив, количество элементов, размер каждого элемента, компаратор )
    // Функция принимает на вход указатель на функцию cmp

    print_array(10, arr);
}
```

Функция-компаратор стандартной функции `qsort` отличается от той, что была написана нами для сортировки городов и звёзд только тем, что она принимает на вход указатели типа `void*`. Это сделано для того, чтобы эта функция была более общей. С помощью неё можно отсортировать как массив чисел, так и массив указателей или массив любых структур. В функции `cmp` нужно привести указатель `void*` к указателю нужного типа.
Задача на стандартную функцию `qsort`:

- Перепишите сортировку звёзд с использованием функции `qsort`.

Как код превращается в последовательность байт:



Из кода на C в код ассемблера:

- Код на языке `C` (`a.c`) переводится в код на языке ассемблера (`a.s`). Эту операцию можно сделать командой

  ```
  gcc -S -masm=intel ./a.c
  ```

- Регистры процессора – это сверхбыстрая память, которая находится внутри процессора. Её размер очень мал(десятки байт), но процессор может доступиться к ней очень быстро (за 1 такт). В примере выше используются 2 регистра: `rbp` и `eax` (`eax` это часть регистра `rax`).

- Процессор может делать множество различных операций. Например, он может переместить некоторое количество байт из одного места в другое. Такие операции называются `mov`. Он может прибавить число (`add`) или умножить на целое (`imull`) и многое другое. `DWORD PTR` просто означает, что операция будет работать с 4-мя байтами.

- В примере выше в регистре `rbp` содержится некоторый адрес. Квадратные скобочки означают разыменование. Поэтому строка

  ```
  mov DWORD PTR [rbp-0x4],0x1234
  ```

  означает, что нужно положить число `0x1234` в 4 байта по адресу `rbp-0x4`

- `mov eax,DWORD PTR [rbp-0x4]`

  означает, что нужно переместить 4 байта, которые хранятся по адресу `rbp-0x4` в регистр `eax`.

- `imull eax,eax,0x7755`

  означает, что нужно умножить содержимое `eax` на `0x7755` и сохранить результат в `eax`.

- `mov  DWORD PTR [rbp-0x4],eax`

  означает, что нужно переместить содержимое `eax` в память по адресу `rbp-0x4`.

- `add  DWORD PTR [rbp-0x4],0x99aa88`

  означает, что нужно добавить к числу по адресу `rbp-0x4` число `0x99aa88`.

- В отличии от кода на языке `C`, код на языке ассемблера различаться на разных процессорах. Код с вычислительной системы одной архитектуры скорей всего не будет работать на другой.

Из кода ассемблера в бинарный код (`.exe`):

- Код на языке ассемблера (`a.s`) переводится в исполняемый файл. Эту операцию можно сделать командой `gcc a.s`
- Каждая операция кодируется некоторым числом, называемым кодом операции (opcode).
- Код операции `mov` на процессорах архитектуры `x86-64` может равняться `c7` или `8b` или `89` или некоторым другим значениям(в зависимости от того куда и откуда мы копируем).
- Например в строке:

  `c7 45 fc 34 12 00 00`

  - `c7` означает, что это операция `mov` (присвоить число переменной в памяти)
  - `45` кодирует регистр `rbp`
  - `fc` кодирует смещение `-0x4`
  - `34 12 00 00` – это 4-х байтовое число `0x1234` (порядок байт – Little Endian)
- `8b 45 fc`

  - `8b` означает, что это операция `mov` (записать число, хранящееся в памяти, в `eax`)
  - `45` кодирует регистр `rbp`
  - `fc` кодирует смещение `-0x4`
- Все коды можно посмотреть тут ref.x86asm.net/coder64.html
- Получается, что в результате компиляции программы код превращается в последовательность байт (инструкций процессора). Эта последовательность байт и хранится в сегменте Текст.
- А указатель на функцию является просто номером первого байта, с которого начинается функция в этом сегменте.
- Менять сегмент Текст во время выполнения программы в большинстве современных операционных систем нельзя.