Домашнее задание: Сортировки

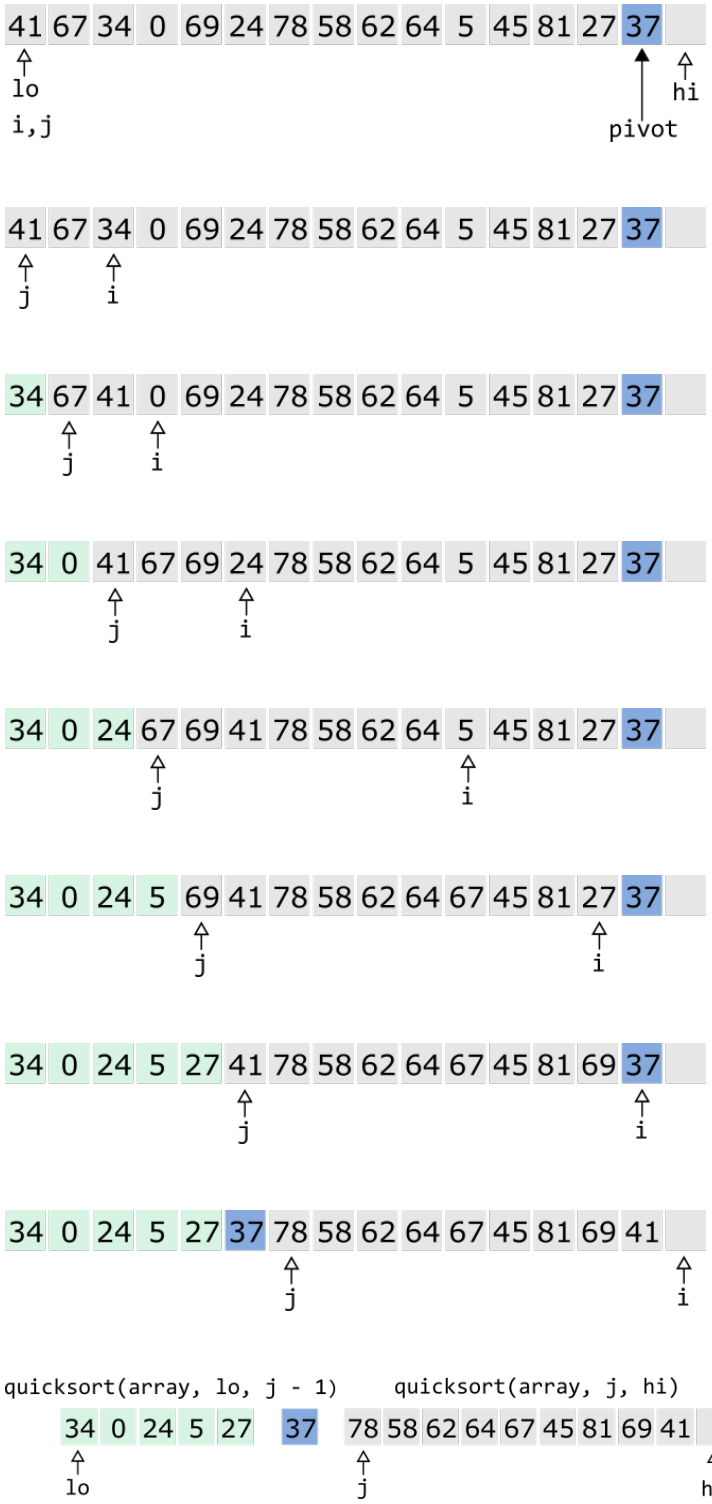
Сложность алгоритмов

Чему равна средняя вычислительная сложность следующих операций?

1. Поиск элемента в 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(самораширяющийся) стек размера N
6. Сортировка выбором массива размера N
7. Сортировка пузырьком массива размера N
8. Быстрая сортировка массива размера N
9. Сортировка подсчётом массива размера N, если максимальный элемент массива равен K
10. Цифровая сортировка массива размера N, если максимальный элемент массива равен K
11. Сортировка Bogosort массива размера N
12. Сортировка слиянием массива размера N
13. Сложение матриц размера $N \times N$
14. Простой алгоритм умножения матриц размера $N \times N$ (строка на столбец)
15. Сложение двух чисел длиной в N цифр (N может быть большим)
16. Простой алгоритм умножения(столбиком) двух чисел длиной в N цифр (N может быть большим)
17. 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)
18. * Добавление элемента в двоичную кучу размера N
19. * Удаление элемента из двоичной кучи размера N

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);
}
```



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.
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 $= -26.7$.
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 $= 4.85$.
6. `spectral_type` - спектральный класс звезды(строка не более чем 15 символов).
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.
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:**
  Создайте массив из структур `Star` подходящего размера и считайте все данные из файла в массив. Файл содержит информацию о 117955 звезде, так что массив нужно создавать в куче (с помощью `malloc`). Для считывания используйте функцию `fscanf` из библиотеки `stdio.h`. Пример считывания:

```c
#include <stdio.h>

int main()
{
    // Открываем файл input.txt на чтение("r"). Для открытия на запись - "w"
    FILE* f = fopen("input.txt", "r");
    fscanf(f, < тут всё то же самое, что и у обычного scanf >)
    // ...
    fclose(f);
}
```

  Учтите, что спецификатор `%s` считывает строку до пробела. Чтобы считать строку до запятой используйте спецификатор `%[^,]` - при этом `s` на конец спецификатора ставить не надо.
  Можно посмотреть на пример в файле `sort_cities.c`.

- **Сохраняем звёзды:**
  Написать функцию `void save_stars(char filename[], Star array[], int n)`, которая будет сохранять города из массива `array` в файл, чьё название хранится в переменной `filename`. Например, при вызове `save_stars(“output.txt”, stars, n);` массив `stars` должен сохраниться в файл `output.txt`.

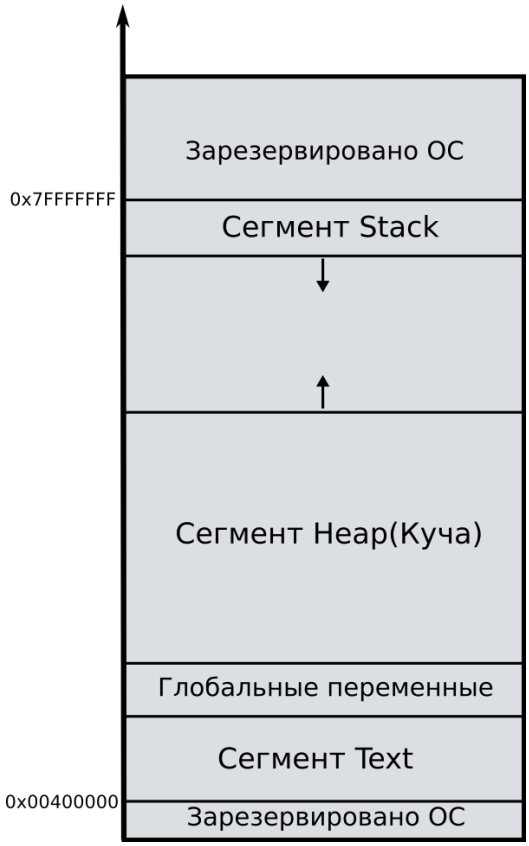
- **Сортировка по видимой с Земли яркости:**
  Создайте функцию `quicksort_magnitude`, чтобы она принимала на вход массив из структур `Star` и сортировала их по возрастанию звёздной величины. Проверьте функцию в `main`, отсортировав структуру и сохранив её в файл `sorted_by_magnitude.txt` с помощью функции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

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

- **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. Проверьте функцию в `main`, отсортировав структуру и сохранив её в файл `sorted_by_temperature.txt`.

- **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`. Можно посмотреть на пример в файле `sort_cities_funcpointer.c`.

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)
   - Быстрее чем куча

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 памятью - гигабайты.
   - Медленней чем стек

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).
   - Адрес функции - адрес первого байта инструкций в этом сегменте.

Пример работы с указателем на функцию:

```
#include <stdio.h>

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

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

Подробней в файле `funcpointer.c`.

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 )

    print_array(10, arr);
}
```

Задачи:

- Задачи от qsort1 до qsort9, кроме qsort6 и qsort7 на judge.mipt.ru, контест “Задание 1.3 (структуры и указатели)”