**ЛЕКЦИЯ 5**

**РАБОТА С ПАМЯТЬЮ**

1. Применение доп. функций. Чтобы функция выглядела корректно для работы функции, нужно, чтобы параметры функции были ожидаемыми.

Пример: метод сортировки массива. Ожидаемые аргументы – сам массив, однако для самой сортировки требуется больше этого. В таком случае делаем перегрузку. Внешняя – которую использует пользователь, и внутренняя – которая реально что – то делает. Внешняя передает нужные параметры во внутреннею;

1. #pragma once – говорит компилятору, что компилируем файл только один раз;
2. Правильный способ печати массива – сначала печатаем первый элемент, потом, начиная со второго, печатаем сначала пробел, а потом элемент. Так у нас не будет лишнего пробела в конце. Имеет смысл проверить, есть ли в массиве вообще элементы. После цикла стоит сделать cout << endl;
3. При сравнении коллекций, следует сначала проверить то, совпадают ли размеры коллекций;
4. Важно уметь использовать флажок в качестве параметра выхода из функции;
5. Машина состояний – массив из функций. Позволяет удобно управлять состоянием программы через индексы (иначе пришлось бы делать switch-case или хуже). Инструмент оптимизации;
6. typedef используется для создание переменной, которая позволяет записать наш основной метод.

**typedef void(\*ptr)(vector<int>&)**

void – возвращаемый тип,

vector<int>& - принимаемый аргумент,

ptr – название типа

Может использоваться, чтобы брать указатели на функцию (делегаты);

Указатель void может использоваться, чтобы сделать ссылку на функцию;

1. Правильно называй свои переменные, не используй служебные слова. Названия должны четко отражать свое назначение;
2. Не использую max, min. Лучше – mx и mn;
3. Шаблоны называют Generic;
4. Не всегда лучше использовать встроенные конструкции;
5. ЧИСТИ ПАМЯТЬ!!!
6. Generic не всегда возможно использовать. Нужно думать о том, с какими типами данных оно не будет работать.
7. system(“chcp 2151 > Null”);
8. while удобно использовать, когда всего одно условие.
9. В начале массив хранит мусор. {0} – пример инициализатора;
10. Важно переносить условия, если они большие;

**РАБОТА С ПАМЯТЬЮ**

1. Ставим красную точку остановки (на каком этапе следует остановится во время выполнения кода);
2. Для начала отладки используем “локальный отладчик”;
3. F10 - построчное прохождение, F11 – построчное прохождение за кодом;
4. Откладка -> окна -> локальные.

**ЕСТЕСТВЕННОЕ СЛИЯНИЕ**

1. Серия – определенное количество элементов заданного множества;
2. Естественное слияние – (2 3 6 17 5) => (236) (17) (5). Находим возврастающие серии. Признак конца серии – следующий элемент меньше предыдущего. Со следующего элемент начинаем организовывать следующую серию.

Пример:

(41, 8, 15, 24, 2, 1, 12, 7) => (41) (8, 15, 24) (2) (1, 12) (7).

**МНОГОФАЗНАЯ СОРТИРОВКА СЛИЯНИЕМ**

1. Исходный массив делится на вспомогательные последовательности (серии);
2. Делим все серии на 2 блока, также делаем 3-ий запасной;
3. Далее мы берем размер меньшей серии , и кол-во элементов, равное этому количеству, перетаскиваем в свободный блок.

Пример:

Блок 1: 13 серий

Блок 2: 8 серий

Блок 3: 0 серий

Из блока 1 и 2 переносим 8 серий в блок 3:

Блок 1: 5 серий

Блок 2: 0 серий

Блок 3: 8 серий

Из блока 1 и блока 3 переносим 5 серий:

Блок 1: 0 серий

Блок 2: 5 серий

Блок 3: 3 серии

1700 элементов.

7(100), 6(100), 4(100), 0 (запас).

Шаг 1:

3(100), 2(100), 0, 4(300)

Шаг 2:

1 (100), 0, 2(500), 2(300)

Шаг 3:

0, 1 (900), 1(500), 1(300)

Формула идеального распределения серий:

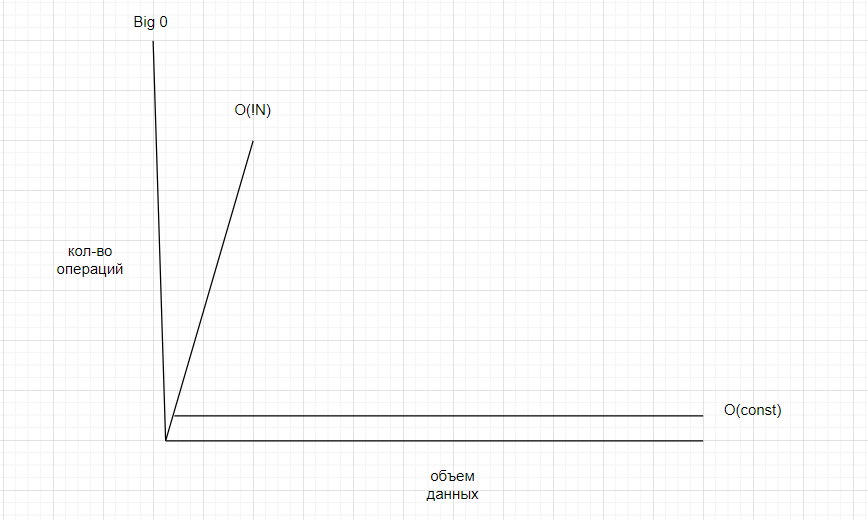
Массив надо раскидать на “ведра”, которые равны двум предыдущем числам в ряду чисел фибоначи;

Число “ведер” вводит юзер.

Если число не ложится в числа фибоначи, делаем фиктивные последовательности.

**ОЦЕНКА СЛОЖНОСТИ АЛГОРИТМОВ, O-БОЛЬШОЕ**

1. Сложность алгоритма оценивается в количестве выполняемых шагов;
2. Кол-во операций зависит от передаваемых параметров;

****

1. Пример:

for (int = 0, i <= 100, i++)

{

//do smt

}

Размер: O(100) = O(const)

for (int = 0, i <= N, i++)

{

//do smt

}

Размер: O(N)

for (int i = 0, i \* i <= N, i++)

{

//do smt

}

Размер: O(sqrt(N))

int a = 1;

for (int i = 0, i \* i <= N, i++)

{

//do smt

}

++a;

Размер: O(1 + N + 1) = O(N)

for (int i = 0, i \* i <= N, i++)

{

//do smt

}

for (int i = 0, i \* i <= N, i++)

{

//do smt

}

Размер: O(N + N) = O(2\*N) = O(N)

for (int i = 0, i \* i <= N, i++)

{

//do smt

}

for (int i = 0, i \* i <= K, i++)

{

//do smt

}

Размер: O(N + K)

for (int i = 0, i \* i <= N, i++)

{

for (int i = 0, i \* i <= K, i++)

{

//do smt

}

}

Размер: O(N \* K)

for (int i = 0, i \* i <= N, i++)

{

for (int i = 0, i \* i <= N, i++)

{

//do smt

}

}

Размер: O(N^2)

1. Принцип значимости аргумента: принимает значение больше чем в 2 раза, чем основное

Пример:

O(3\*N + N^2 + 2^N) = O(2^N)

1. Логарифмический O(logN):

Основание логарифма – 2

Пример: бинарный поиск

1. Задачи, с полным перебором всех вариантов – O(N!).

**НЕМНОГО О ПОИСКАХ**

1. И бинарный, и интерполярный требуют сортировку, второй быстрее первого.

**ПОИСК ПОДСТРОКИ В СТРОКЕ МЕТОДОМ ПРЯМОГО ПОИСКА**

1. Посимвольно передвигаем строку в массив до тех пор, пока не получим совпадающие строки;
2. Очень прост, но время выполнения может вырасти до неадекватных значений;