Белорусский государственный технологический университет

Факультет информационных технологий

Кафедра программной инженерии

 Реферат

По дисциплине «Основы алгоритмизации и программирования»

На тему «Нотации Big O для оценки алгоритма»

 Выполнила:

Студентка1 курса 6 группы

Литвинчук Дарья Валерьевна

Преподаватель: доц Белодед Н.И.

2024, Минск

**Примеры нотаций Big O:**

**1)O(1): Константная сложность.**

Время выполнения алгоритма не зависит от размера входных данных.

Алгоритмы с константной сложностью являются наиболее эффективными с точки зрения времени выполнения, так как они требуют фиксированного числа операций, независимо от размера входных данных. Примеры алгоритмов с константной сложностью включают доступ к элементу массива по индексу, выполнение арифметических операций и проверку условий с помощью одного оператора.

Однако, стоит отметить, что константная сложность не означает, что алгоритм всегда будет выполняться мгновенно. Время выполнения может быть ограничено другими факторами, такими как аппаратные ограничения или ввод/вывод операций. Однако, само время выполнения алгоритма будет постоянным и не будет зависеть от размера входных данных.

val nums = intArray(1, 2, 3, 4, 5)

val firstNumber = nums[0]

**2)O(n): Линейная сложность.**

Время выполнения алгоритма пропорционально размеру входных данных.

Алгоритмы с линейной сложностью выполняют операции для каждого элемента входных данных. Например, если у нас есть массив длины n, то алгоритм с линейной сложностью выполнит операции примерно n раз.

fun sum(n: Int) : Int{

if (n == 1) return 1

return n + sum(n - 1)

}

**3)O(log n): Логарифмическая сложность.**

время выполнения алгоритма растёт логарифмически по отношению к размеру входных данных.

Алгоритмы с логарифмической сложностью обычно связаны с делением пространства поиска на половины на каждом шаге. Они часто применяются в контексте бинарного поиска или поиска в сортированных структурах данных.

int binarSearch(int arr[], int Value, int low, int high) {

if (low > high)

return -1;

int mid = low + (high - low) / 2;

if (arr[mid] == Value)

return mid;

else if (arr[mid] < Value)

return binarSearch(arr, Value, mid + 1, high);

else

return binarSearch(arr, Value, low, mid - 1);

}

int main() {

int myArray[] = { 1, 2, 3, 4, 5, 6 };

int Value = 6;

int arraySize = sizeof(myArray) / sizeof(myArray[0]);

int index = binarSearch(myArray, Value, 0, arraySize - 1);

}

**4)O(n log n): Линейно-логарифмическая сложность.**

Время выполнения алгоритма растет быстрее, чем линейно, но медленнее, чем квадратично.

Линейно-логарифмические алгоритмы обычно комбинируют линейные и логарифмические операции. Они часто встречаются в задачах, где требуется разбиение пространства поиска на подзадачи, а затем объединение их результатов.

Например, сортировка слиянием (merge sort).

void merge(int\* arr, int left, int mid, int right) {

int n1 = mid - left + 1;

int n2 = right - mid;

int\* L = new int[n1];

int\* R = new int[n2];

for (int i = 0; i < n1; i++)

L[i] = arr[left + i];

for (int j = 0; j < n2; j++)

R[j] = arr[mid + 1 + j];

int i = 0;

int j = 0;

int k = left;

while (i < n1 && j < n2) {

if (L[i] <= R[j]) {

arr[k] = L[i];

i++;

}

else {

arr[k] = R[j];

j++;

}

k++;

}

while (i < n1) {

arr[k] = L[i];

i++;

k++;

}

while (j < n2) {

arr[k] = R[j];

j++;

k++;

}

delete[] L; delete[] R;

}

void mergeSort(int\* arr, int left, int right) {

if (left < right) {

int mid = left + (right - left) / 2;

mergeSort(arr, left, mid);

mergeSort(arr, mid + 1, right);

merge(arr, left, mid, right);

}

}

**5)O(n^2): Квадратичная сложность.**

Время выполнения алгоритма зависит от квадрата размера входных данных.

Квадратичная сложность часто возникает в алгоритмах, которые содержат два вложенных цикла, где каждый цикл выполняет итерации, зависящие от размера входных данных. При таком подходе каждый элемент первого цикла будет сравниваться с каждым элементом второго цикла, что приводит к O(n^2) операций.

Например, сортировка пузырьком (bubble sort).

void sortBubble(int\* arr, int N){

int buff = 0;

for (int i = 0; i < N - 1; i++){

for (int j = N - 1; j > i; j--){

if (arr[j] < arr[j - 1]){

buff = arr[j - 1];

arr[j - 1] = arr[j];

arr[j] = buff;

}

}

}

}