**Лабораторная работа № 3.**

**Оптимизация временной сложности алгоритма**

**Цели работы**

1. Изучение основных понятий, связанных с оценкой временной сложности алгоритмов .

2. Изучение практических приемов оптимизации кода.

**1. Основные понятия**

В информатике временна́я сложность алгоритма характеризует время его работы как функцию от размерности (объема) исходных данных. Временная сложность алгоритма обычно выражается с использованием нотации »*O*» большое, которая исключает коэффициенты и члены меньшего порядка. Если сложность выражена таким способом, говорят об *асимптотическом* описании временной сложности, т.е. при стремлении размера входа к бесконечности. Например, если время, которое нужно алгоритму для выполнения работы, для всех входных данных длины *n* не превосходит 5*n*3 + 3*n* для некоторого *n* , асимптотическая временная сложность равна *O*(*n*3).

Самым простым методом оценки временной сложности является подсчёт количества элементарных операций, осуществляемых алгоритмом, в предположении, что элементарная операция занимает для выполнения фиксированное время. Тогда полное время выполнения и число элементарных операций, выполненных алгоритмом, отличаются не более, чем на постоянный множитель.

Основные типы сложности по времени представлены в табл. 1

Таблица 1. Временная сложность алгоритмов

|  |  |  |
| --- | --- | --- |
| Название | Оценка времени работы | Пример алгоритма |
| постоянное (константное) время | *O*(1) | Определение чётности целого числа |
| логарифмическое время | *O*(log *n*) | Метод деления пополам (бисекции) |
| линейное время | *O(n)* | Поиск максимального или минимального элемента в массиве |
| квадратичное время | *O*(*n*2) | Выбор всех элементов квадратной матрицы размерности n\*n |
| экспоненциальное время | 2*O*(*n*) | Решение задачи коммивояжёра методами линейного программирования |
| факториальное время | *O*(*n*!) | Перебор всех комбинаций из *N* заданных значений |

**2. ЗАДАНИЕ К ЛАБОРАТОРНОЙ РАБОТЕ**

1. В соответствии с вариантом задания, реализовать алгоритмы так, чтобы их временная сложность соответствовала требуемой в постановке задачи или были проведены иные оптимизации (см. табл. 2). Решения представить в виде работающих программ на C++.

Таблица 2. Варианты задания

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| № | Алгоритм | Реализация на C++ | Текущая временная сложность кода | Требуемая временная сложность кода и/или улучшение алгоритма |
| 1 | Выбор элементов главной диагонали квадратной матрицы A[n,n] | for (int i=0; i<n; i++)  for (int j=0;j<n; j++)  if (i==j)  cout << a[i][j] << " "; | *O*(*n*2) | *O*(*n*) |
| 2 | Сортировка элементов вектора a[n] по возрастанию | for (int i=0; i<n; i++)  for (int j=0;j<n; j++)  if (a[i]>a[j]) {  int t=a[i]; a[i]=a[j];  a[j]=t;  } | *O*(*n*2) | *O*(*n*2/2) |
| 3 | Поиск номера максимального элемента в массиве a[n] | int max=a[0], imax=0;  for (int i=1; i<n; i++)  if (a[i]>max) {  max = a[i]; imax=i;  } | *O*(*n*) | *O*(*n*) (*реализовать алгоритм без применения 2 дополнит. переменных*) |
| 4 | Обработка всех элементов симметрической относительно главной диагонали квадратной матрицы A[n,n] | for (int i=0; i<n; i++)  for (int j=0;j<n; j++)  if (i==j) cout <<  a[i][j] << a[j][i]  << enl; | *O*(*n*2) | *O*(*n*2/2) |
| 5 | Определение количества минимальных элементов в целочисленном массиве A[n] | int min=a[0], k=0;  for (int i=1; i<n; i++)  if (a[i]<min) min=a[i];  for (int i=0; i<n; i++)  if (a[i]==min) k++; | *O(2\*n)* | *O*(*n*) |
| 6 | Поиск суммы элементов последовательности n , 2\*n, …, m\*n | double s=0; int n=3, m=10;  for (int i=1; i<=m; i++)  s+=i\*n; | *O(n)* | *O(1)* |
| 7 | Вычисление суммы степенного ряда 1+x+x2+…+xn | double sum=0, x=0.5;  for (int i=0; i<=n; i++)  sum += pow(x,i); | *O*(*n*) | *O*(*n*) *(реализовать алгоритм без необходимости вызова функции pow на каждом шаге цикла)* |
| 8 | Получение четырёхзначного натурального числа из пары двузначных | int a=12,b=34;  char buf[4];  itoa(a,buf,10);  itoa(b, &buf[2], 10);  cout << buf; | *O(1)* | *O(1)* *(устранить использование дополнительного строкового буфера)* |
| 9 | Поиск в матрице натуральных чисел, строки и столбцы которой упорядочены по возрастанию, элемента со значением find | const int n=4,m=3;  int a[n][m]={  {10,20,30},{15,35,40},  {20,55,60},{40,80,90}  };  int find = 55;  bool found = false;  for (int i=0; i<n; i++)  for (int j=0; j<m; j++)  {  if (a[i][j]==find) {  found = true; break;  }  } | *O*(*n*2) | *Меньше, чем O*(*n*2)*, желательно, логарифмическое* |
| 10 | Поиск в массиве натуральных чисел a[n] пары элементов со значениями item1 и item2, расстояние между которыми минимально | const int n=10;  int a[n] = {  3,5,10,8,4,12,5,3,0,7  };  int item1=3,item2=8,min=n;  for (int i=0; i<n; i++)  for (int j=0; j<n; j++)  {  if (a[i]==item1 &&  a[j]==item2 &&  abs(i-j)<min) {  min = abs(i-j);  }  }  cout << min; | *O*(*n*2) | *O*(*n*) |

2. Составить отчет по лабораторной работе. Отчет должен содержать следующее.

**1.** Титульный лист.

**2.** Цель лабораторной работы.

**3.** Вариант задания.

**4.** Полный компилируемый листинг реализованной программы.

**5.** Вывод по результатам лабораторной работы.

**3. ВАРИАНТЫ ЗАДАНИЙ**

Варианты заданий взять из табл. 2.

Таблица 2. Варианты заданий

|  |  |  |
| --- | --- | --- |
| № вар. | Алгоритм 1 | Алгоритм 2 |
| 1 | 1 | 10 |
| 2 | 2 | 9 |
| 3 | 3 | 8 |
| 4 | 4 | 7 |
| 5 | 5 | 6 |
| 6 | 6 | 5 |
| 7 | 7 | 4 |
| 8 | 8 | 3 |
| 9 | 9 | 2 |
| 10 | 10 | 1 |

**4. КОНТРОЛЬНЫЕ ВОПРОСЫ**

1. Что характеризует понятие временной сложности алгоритма?
2. Что является основным методом оценки временной сложности алгоритма?
3. Какие существуют типы сложности по времени работы алгоритма?
4. Всегда ли возможно улучшение временной сложности?
5. Что означают понятия "алгоритм сложности *O*(*n*)", "алгоритм сложности *O*(1)"?