Пензенский государственный университет

Кафедра "Вычислительная техника"

**ОТЧЕТ**

по дисциплине "Логика и основы алгоритмизации в инженерных задачах"

по лабораторной работе №2

на тему "Оценка времени выполнения программ"

Выполнил студент гр. 22ВВС1:

Беспалов Н.А.

Проверили:

Юрова О.В.

Акифьев И.В.

Пенза 2024

**Цель работы**: Научиться оценивать порядок сложности программы, время работы функций и алгоритмов на заданной программе.

**Практическое задание:**

**Задание 1:**

1. Вычислить порядок сложности программы (*О*-символику).
2. Оценить время выполнения программы и кода, выполняющего перемножение матриц, используя функции библиотеки time.h для матриц размерами от 100, 200, 400, 1000, 2000, 4000, 10000.
3. Построить график зависимости времени выполнения программы от размера матриц и сравнить полученный результат с теоретической оценкой.

**Задание 2**:

1. Оценить время работы каждого из реализованных алгоритмов на случайном наборе значений массива.
2. Оценить время работы каждого из реализованных алгоритмов на массиве, представляющем собой возрастающую последовательность чисел.
3. Оценить время работы каждого из реализованных алгоритмов на массиве, представляющем собой убывающую последовательность чисел.
4. Оценить время работы каждого из реализованных алгоритмов на массиве, одна половина которого представляет собой возрастающую последовательность чисел, а вторая, – убывающую.
5. Оценить время работы стандартной функции qsort, реализующей алгоритм быстрой сортировки на выше указанных наборах данных.

**Ход работы:**

**Задание 1.**

1. О(N^3)
2. 100 – 0,003 c

200 – 0,0034 с

400 – 0,186 с

1000 – 3,7 с

2000 – 43,2 с

4000 – 387,4 с

10000 – 4300,675 с

1. Построили график:

Зависимость времени от размера матриц

Листинг:

#include <stdio.h>

#include <stdlib.h>

#include <time.h>

unsigned int a[100][100], b[100][100], c[100][100], elem\_c;

int main(void)

{

setvbuf(stdin, NULL, \_IONBF, 0);

setvbuf(stdout, NULL, \_IONBF, 0);

clock\_t start = clock();

int i = 0, j = 0, r;

int k = 100;

srand(time(NULL));

while (i < k)

{

while (j < k)

{

a[i][j] = rand() % 100 + 1;

j++;

}

i++;

}

srand(time(NULL));

i = 0; j = 0;

while (i < k)

{

while (j < k)

{

b[i][j] = rand() % 100 + 1;

j++;

}

i++;

}

for (i = 0;i < k;i++)

{

for (j = 0;j < k;j++)

{

elem\_c = 0;

for (r = 0;r < k;r++)

{

elem\_c = elem\_c + a[i][r] \* b[r][j];

c[i][j] = elem\_c;

}

}

}

clock\_t end = clock();

double seconds = (double)(end - start) / CLOCKS\_PER\_SEC;

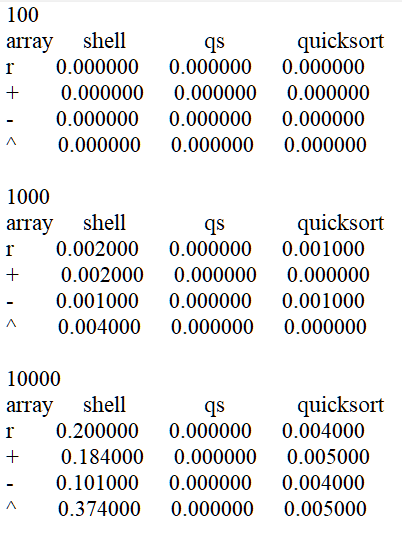
printf("Program time is - %f", seconds);

return(0);

}

**Задание 2.**

**Результат работы:**

****

Листинг:

#include <stdio.h>

#include <stdlib.h>

#include <time.h>

void random(int\*a,int k) {

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

{

a[i] = rand() % k + 1;

}

}

void narost(int\*b,int k) {

for (int i = 1;i < k;i++)

{

b[i] = b[i - 1] + 1;

}

}

void nerost(int\*c,int k) {

for (int i = 1;i < k;i++)

{

c[i] = c[i - 1] - 1;

}

}

void triada(int\*d,int k) {

for (int i = 1;i < k;i++)

{

if (i < (k / 2 + 1))

d[i] = d[i - 1] + 1;

else

d[i] = d[i - 1] - 1;

}

}

void shell(int\* items, int count)

{

int i, j, x, gap, k;

for (k = 0; k < count; k++) {

gap = items[k];

for (i = gap; i < count; ++i) {

x = items[i];

for (j = i - gap; (x < items[j]) && (j >= 0); j = j - gap)

items[j + gap] = items[j];

items[j + gap] = x;

}

}

}

void qs(int\* items, int left, int right) //вызов функции: qs(items, 0, count-1);

{

int i, j;

int x, y;

i = left; j = right;

x = items[(left + right) / 2];

do {

while ((items[i] < x) && (i < right)) i++;

while ((x < items[j]) && (j > left)) j--;

if (i <= j) {

y = items[i];

items[i] = items[j];

items[j] = y;

i++; j--;

}

} while (i <= j);

if (left < j) qs(items, left, j);

if (i < right) qs(items, i, right);

}

int comp(const void\* i, const void\* j)

{

return \*(int\*)i - \*(int\*)j;

}

int main() {

int k = 100;

double shel[4],qsrt[4],qsrt1[4];

clock\_t start, end;

srand(time(NULL));

int\* a;

int\* b;

int\* c;

int\* d;

FILE\* f;

char name[] = "Sort.txt";

fopen\_s(&f,name, "a+");

for (int si = 0;si < 3;si++)

{

a = (int\*)malloc(k \* sizeof(int));

b = (int\*)malloc(k \* sizeof(int));

c = (int\*)malloc(k \* sizeof(int));

d = (int\*)malloc(k \* sizeof(int));

b[0] = 1;

c[0] = k;

d[0] = 1;

random(a, k);

start = clock();

shell(a, k);

end = clock();

shel[0] = (double)(end - start) / CLOCKS\_PER\_SEC;

random(a, k);

start = clock();

qs(a, a[0], a[1]);

end = clock();

qsrt[0] = (double)(end - start) / CLOCKS\_PER\_SEC;

random(a, k);

start = clock();

qsort(a, k, sizeof(int), comp);

end = clock();

qsrt1[0] = (double)(end - start) / CLOCKS\_PER\_SEC;

narost(b, k);

start = clock();

shell(b, k);

end = clock();

shel[1] = (double)(end - start) / CLOCKS\_PER\_SEC;

random(b, k);

start = clock();

qs(b, b[0], b[1]);

end = clock();

qsrt[1] = (double)(end - start) / CLOCKS\_PER\_SEC;

random(b, k);

start = clock();

qsort(b, k, sizeof(int), comp);

end = clock();

qsrt1[1] = (double)(end - start) / CLOCKS\_PER\_SEC;

nerost(c, k);

start = clock();

shell(c, k);

end = clock();

shel[2] = (double)(end - start) / CLOCKS\_PER\_SEC;

random(c, k);

start = clock();

qs(c, c[0], c[1]);

end = clock();

qsrt[2] = (double)(end - start) / CLOCKS\_PER\_SEC;

random(c, k);

start = clock();

qsort(c, k, sizeof(int), comp);

end = clock();

qsrt1[2] = (double)(end - start) / CLOCKS\_PER\_SEC;

triada(d, k);

start = clock();

shell(d, k);

end = clock();

shel[3] = (double)(end - start) / CLOCKS\_PER\_SEC;

random(d, k);

start = clock();

qs(d, d[0], d[1]);

end = clock();

qsrt[3] = (double)(end - start) / CLOCKS\_PER\_SEC;

random(d, k);

start = clock();

qsort(d, k, sizeof(int), comp);

end = clock();

qsrt1[3] = (double)(end - start) / CLOCKS\_PER\_SEC;

fprintf(f,"%d\n", k);

fprintf(f, "shell qs quicksort\n");

for (int i = 0;i < 4;i++) {

if (i == 0)

fprintf(f, "r %f %f %f\n", shel[i],qsrt[i],qsrt1[i]);

if (i == 1)

fprintf(f, "+ %f %f %f\n", shel[i], qsrt[i], qsrt1[i]);

if (i == 2)

fprintf(f, "- %f %f %f\n", shel[i], qsrt[i], qsrt1[i]);

if (i == 3)

fprintf(f, "^ %f %f %f\n", shel[i], qsrt[i], qsrt1[i]);

}

fprintf(f,"\n", k);

k \*= 10;

free(a);

free(b);

free(c);

free(d);

}

fclose(f);

}

**Вывод**:

Мы научились оценивать сложность работы программ, время работы функций и алгоритмов. Сложность представленной программы О(n^3). Оценив время выполнения программы на матрицах разных размеров, мы построили график зависимости и пронаблюдали резкое возрастание времени с увеличением размера массива.

Мы оценили время работы алгоритмов shell, qs, quicksort на заданных алгоритмах. В итоге, на массивах размеров 100 время выполнения алгоритмов стремилось к нулю. На массивах размером 1000 алгоритм qs так же работал почти мгновенно, quicksort дольше работал над рандомным массивом и убывающем. Shell дольше всего работал над треугольном массивом и меньше над убывающим. На массивах размером 10000 время работы qs стремилось к 0, время работы quicksort отличалось на минимальное значение, а shell дольше всего работал над треугольном массивом и меньше над убывающим.