Ministerul Educaţiei, Culturii și Cercetării

UNIVERSITATEA TEHNICĂ A MOLDOVEI

Departamentul Ingineria Software și Automatică

RAPORT

La structuri de date și algoritmi

Lucrarea de laborator nr. 8

Varianta 18

A efectuat:

st. gr. TI-206 Cătălin Pleșu

A verificat:

Lector universitar Vitalie Mititelu

Chișinău 2021

Tema: Analiza empirică a algoritmilor de sortare şi de căutare

Scopul: Studierea posibilităţilor şi mijloacelor limbajului C de programare a algoritmilor de sortare şi de căutare pentru tablouri unidimensionale şi obţinerea deprinderilor de analiză empirică a algoritmilor

Sarcina: Să se scrie un program în limbajul C pentru analiza empirică a algoritmului propus (după variantă) cu crearea funcţiei de căutare sau sortare pentru tabloul unidimensional de n elemente cu afişarea la ecran a următorului meniu de opţiuni:

1. Tabloul demonstrativ de n elemente (5<= n <=20).
2. Tabloul cu valori aleatorii (n=10000, n=100000, n=1000000).
3. Tabloul sortat crescător (n=10000, n=100000, n=1000000).
4. Tabloul sortat descrescător (n=10000, n=100000, n=1000000).
5. Ieşire din program.

Analiza empirică a algoritmului constă în:

(a) determinarea timpului de rulare, numărului de comparaţii, numărului de schimbări (sau deplasări) de elemente pentru tablouri cu 3 diferite seturi de valori şi cu 3 diferite volume de elemente;

(b) compararea şi analiza rezultatelor obţinute utilizând funcţia creată şi funcţia din biblioteca standard a limbajului C;

(c) tragerea concluziilor.

Varianta 18: Sortare Shell în ordine descendentă.

Rezumat la temă:

ShellSortă: este în principal o variantă a sortării prin inserție. În sortarea prin inserție, mutăm elementele cu o singură poziție înainte. Când un element trebuie deplasat cu mult înainte, sunt implicate multe mișcări. Ideea shellSort este de a permite schimbul de articole îndepărtate. În shellSort, realizăm matricea h-sortată pentru o valoare mare de h. Continuăm să reducem valoarea lui h până când devine 1. Se spune că o matrice este sortată în h dacă se sortează toate sublistele fiecărui element h'th. Urmează implementarea ShellSort.

Qsort: Biblioteca standard C oferă qsort () care poate fi folosit pentru sortarea unui tablou. După cum sugerează și numele, funcția folosește algoritmul QuickSort pentru a sorta matricea dată. Următorul este prototipul qsort ()

Punctul cheie despre qsort () este comparatorul funcției de comparare. Funcția de comparare ia două argumente și conține logică pentru a decide ordinea lor relativă în ieșire sortată. Ideea este de a oferi flexibilitate, astfel încât qsort () să poată fi utilizat pentru orice tip (inclusiv tipurile definite de utilizator) și să poată fi utilizat pentru a obține orice ordine dorită (crescătoare, descrescătoare sau orice alta). Funcția de comparare ia două indicatoare ca argumente (ambele tipate la const void *) și definește ordinea elementelor prin returnare (într-un mod stabil și tranzitiv

Algorithm	Time Complexity			Space Complexity
	Best	Average	Worst	Worst
Quicksort	O(n log(n))	O(n log(n))	O(n^2)	O(log(n))
Mergesort	O(n log(n))	O(n log(n))	O(n log(n))	O(n)
Timsort	O(n)	O(n log(n))	O(n log(n))	O(n)
Heapsort	O(n log(n))	O(n log(n))	O(n log(n))	O(1)
Bubble Sort	O(n)	O(n^2)	O(n^2)	O(1)
Insertion Sort	O(n)	O(n^2)	O(n^2)	O(1)
Selection Sort	O(n^2)	O(n^2)	O(n^2)	O(1)
Shell Sort	O(n)	O((nlog(n))^2)	O((nlog(n))^2)	O(1)
Bucket Sort	O(n+k)	O(n+k)	O(n^2)	O(n)
Radix Sort	O(nk)	O(nk)	O(nk)	O(n+k)

Cod sursă:

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <math.h>

long long int comparatii = 0;
long long int schimbari = 0;

int *genereaza_mi_tabloul(int n)
{
srand(time(NULL));
int *tablou = (int *)malloc(sizeof(int) * n);
for (int i = 0; i < n; i++)
{
tablou[i] = rand() % 20001 - 10000;
}
return tablou;
}
void printeaza_mi_tablou(int *tablou, int n)
{
for (int i = 0; i < n; i++)
printf("%d ", tablou[i]);
printf("\n");
}
void shell_sorteaza_mi_descrescator(int *tablou, int n)
{
comparatii = 0;
schimbari = 0;
for (int interval = n / 2; interval > 0; interval /= 2)
{
for (int i = interval; i < n; i += 1)
{
++comparatii;
int temp = tablou[i];
int j;
for (j = i; j >= interval && tablou[j - interval] < temp; j -= interval, ++comparatii)
{

tablou[j] = tablou[j - interval];
++schimbari;
}
tablou[j] = temp;
}
}
}
int comparare(const void *a, const void *b)
```

```
{
++comparatii;
return (*(int *)b - *(int *)a);
}
int *duplicare_mi_tabloul(int *tablou, int n)
{
int *duplicare = (int *)malloc(sizeof(int) * n);
for (int i = 0; i < n; i++)
{
duplicare[i] = tablou[i];
}
return duplicare;
}
void compara_sortarea(int *arr, int n)
{
int *arr2 = duplicare_mi_tabloul(arr, n);
clock_t start, sfarsit;
printf("########## N = %d | 10^%d ##########\n", n, (int)log10(n));
printf("SHELL SORT:\n");
start = clock();
shell_sorteaza_mi_descrescator(arr, n);
sfarsit = clock();
printf("timpului de rulare: %f secunde\n", (double)(sfarsit - start) / CLOCKS_PER_SEC);
printf("comparatii: %lld\n", comparatii);
printf("schimbari: %lld\n", schimbari);
printf("QSORT (stdlib)\n");
comparatii = 0;
start = clock();
qsort(arr2, n, sizeof(int), comparare);
sfarsit = clock();
printf("timpului de rulare: %f secunde\n", (double)(sfarsit - start) / CLOCKS_PER_SEC);
printf("comparatii: %lld\n", comparatii);
printf("\n");
}
int main()
{
printf("======================\n");
printf("60 sec = 1 min\n");
printf("0.001 sec = 1 milisec\n");
printf("======================\n");
srand(time(NULL));
int *arr = NULL, n = (rand() % 16) + 5;
arr = genereaza_mi_tabloul(n);
int *arr2 = duplicare_mi_tabloul(arr, n);
printf("################################\n");
printf("Tabloul demonstrativ de %d elemente\n", n);
printf("tablou initial:\n");
printeaza_mi_tablou(arr, n);
```

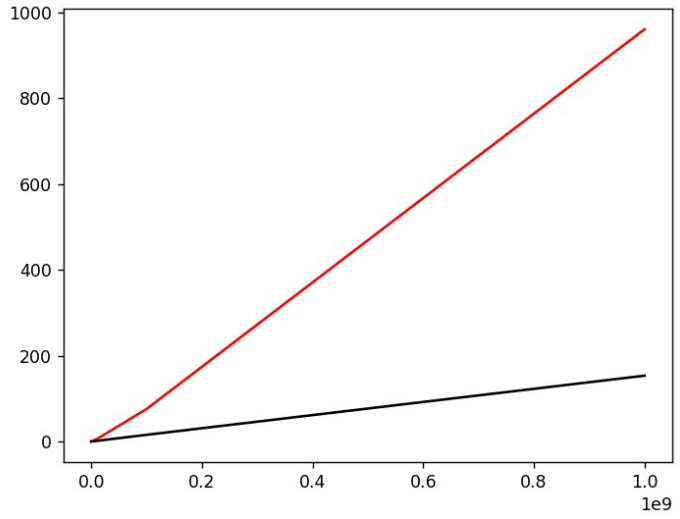
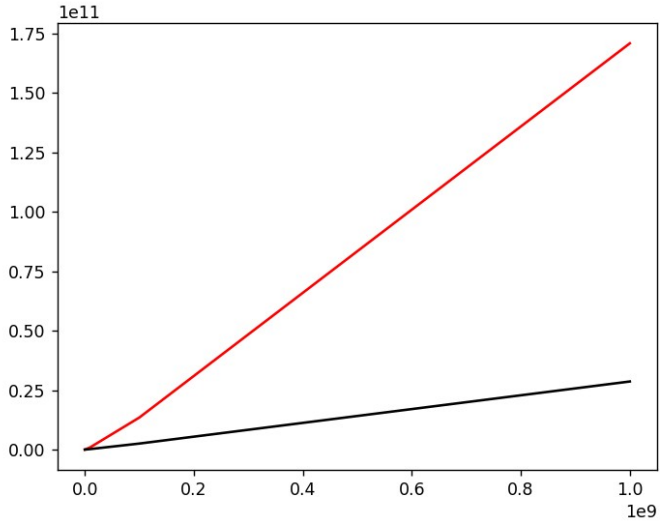
```
printf("#################################\n");
printf("SHELL SORT:\n");
shell_sorteaza_mi_descrescator(arr, n);
printeaza_mi_tablou(arr, n);
printf("timpului de rulare: nesemnificativ\n");
printf("comparatii: %lld\n", comparatii);
printf("schimbari: %lld\n", schimbari);
printf("#################################\n");
printf("QSORT (stdlib):\n");
comparatii = 0;
qsort(arr2, n, sizeof(int), comparare);
printeaza_mi_tablou(arr2, n);
printf("timpului de rulare: nesemnificativ\n");
printf("comparatii: %lld\n", comparatii);

int N[] = {10000, 100000, 1000000, 10000000, 100000000, 1000000000};
for (int i = 0; i < sizeof(N) / sizeof(int); i++)
{
free(arr);
arr = genereaza_mi_tabloul(N[i]);
compara_sortarea(arr, N[i]);
}
return EXIT_SUCCESS;
}
```

Testarea programului (rezultatul executării):

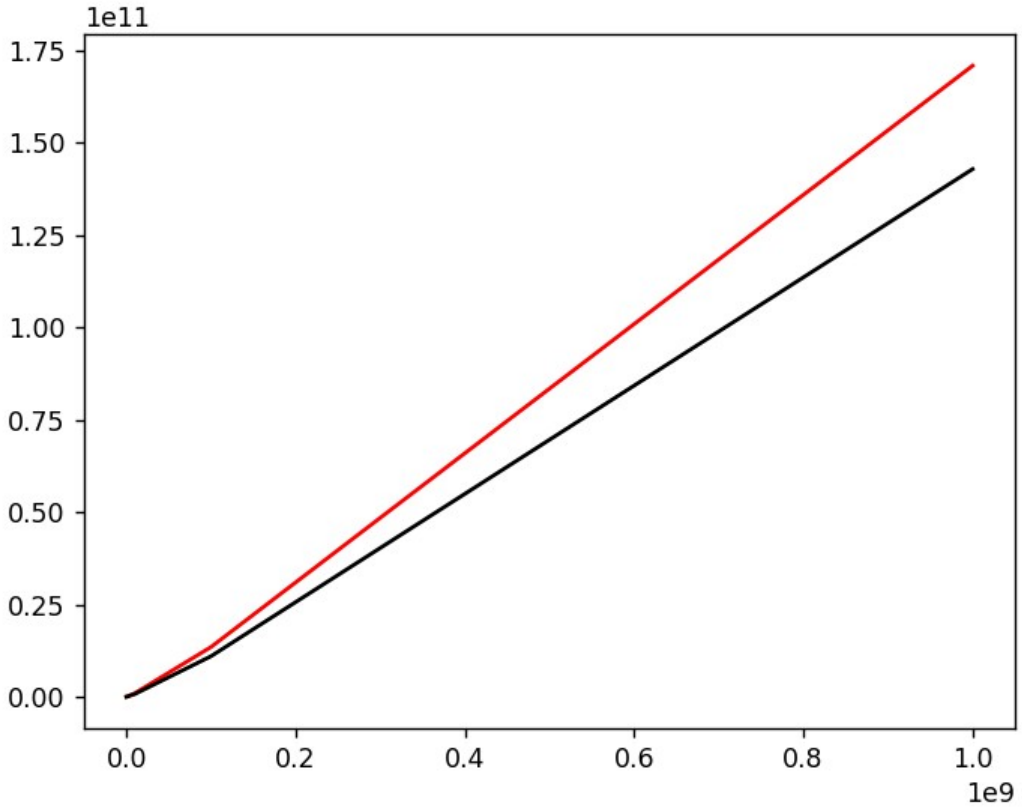
```
======================
60 sec = 1 min
0.001 sec = 1 milisec
======================
################################
Tabloul demonstrativ de 6 elemente
tablou initial:
8513 -8405 -691 3949 6351 9437
################################
SHELL SORT:
9437 8513 6351 3949 -691 -8405
timpului de rulare: nesemnificativ
comparatii: 13
schimbari: 5
################################
QSORT (stdlib):
9437 8513 6351 3949 -691 -8405
timpului de rulare: nesemnificativ
comparatii: 9
########## N = 10000 | 10^4 ##########
SHELL SORT:
timpului de rulare: 0.001923 secunde
comparatii: 271022
schimbari: 151017
QSORT (stdlib)
timpului de rulare: 0.000997 secunde
comparatii: 120407

########## N = 100000 | 10^5 ##########
SHELL SORT:
timpului de rulare: 0.026895 secunde
comparatii: 4257957
schimbari: 2757951
QSORT (stdlib)
timpului de rulare: 0.011903 secunde
comparatii: 1536323
```

```
########## N = 1000000 | 10^6 ##########
SHELL SORT:
timpului de rulare: 0.406007 secunde
comparatii: 70645885
schimbari: 52645878
QSORT (stdlib)
timpului de rulare: 0.131649 secunde
comparatii: 18674494

########## N = 10000000 | 10^7 ##########
SHELL SORT:
timpului de rulare: 5.441267 secunde
comparatii: 967482034
schimbari: 747482026
QSORT (stdlib)
timpului de rulare: 1.391839 secunde
comparatii: 220097512

########## N = 100000000 | 10^8 ##########
SHELL SORT:
timpului de rulare: 75.640205 secunde
comparatii: 13479076439
schimbari: 10979076427
QSORT (stdlib)
timpului de rulare: 15.553826 secunde
comparatii: 2532910485

########## N = 1000000000 | 10^9 ##########
SHELL SORT:
timpului de rulare: 960.657452 secunde
comparatii: 170880727195
schimbari: 142880727182
QSORT (stdlib)
timpului de rulare: 153.375264 secunde
comparatii: 28642535174
```

Vizualizarea datelor: sehll sort ; qsort
interval elemente de la 10^4 la 10^9

timpul de executare in secunde	numărul de comparaţii efectuate
	

Numărul de comparaţii şi substituiri pentru shell sort



In urma analizării rezultatelor obținute pot spune: cu, cât numărul de elemente este mai mic cu atât diferența dintre timpul de execuție este mai mică, deşi asta este de aşteptat.
Pentru primul tablou ce avea 10^4 elemente timpul de executare a fost în jur de două ori mai pentru sortarea oferită de algoritmul dîn librăria standard, iar pentru ultimul tablou de 10^9 elemente diferența a fost de de peste 6 ori deci cu fiecare element adăugat eficiența algoritmului din qsort crește.

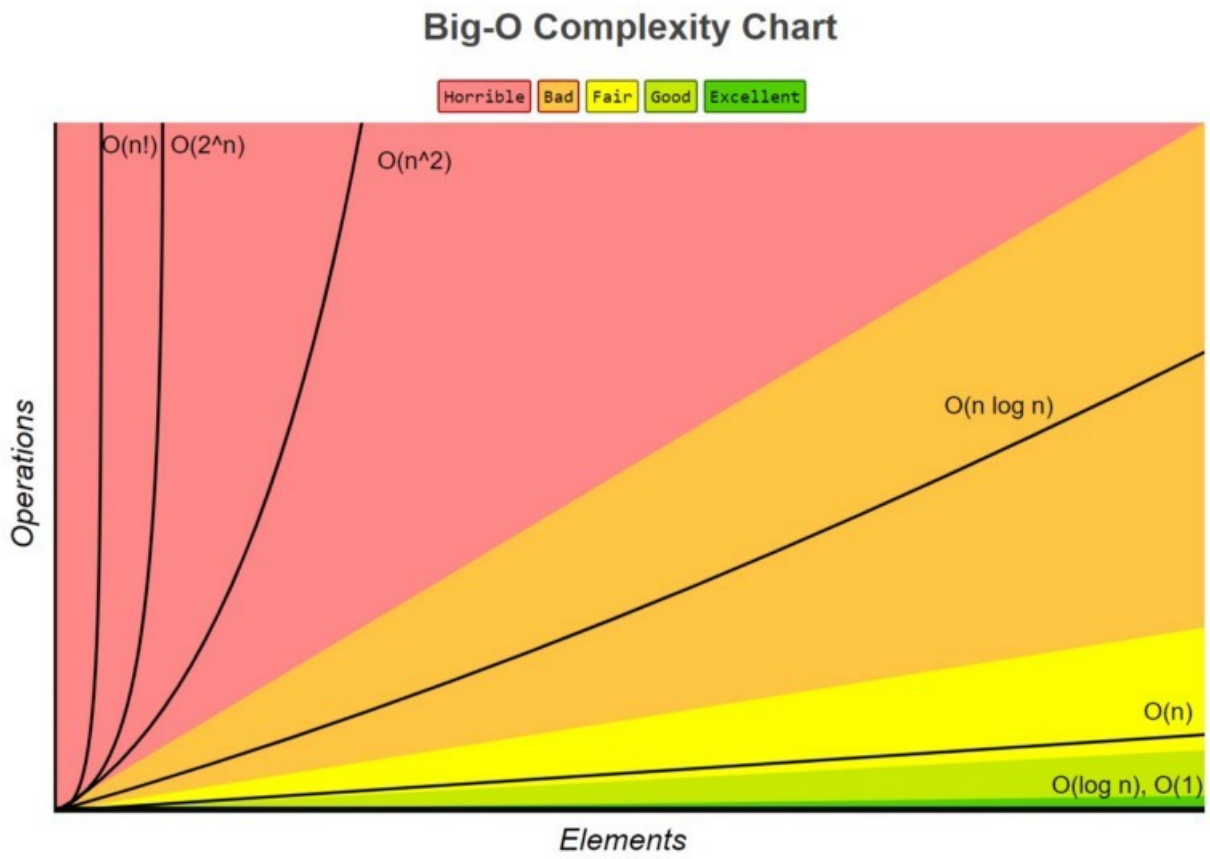
Concluzii:

In urma efectuării acestui laborator am învățat un algoritm nou de sortare (shell sort) și am aplicat pentru prima data qsort despre care deja știam. Am generat niște date pe care apoi le-am reprezentat grafic cu ajutorul unui script in python.

Am ajuns la concluzia că funcția de sortare din librăria standard este mai eficientă decât shell sort, deci în caz de necesitate de a sorta ceva voi opta pentru qsort.

Big O notation pentru shell sort este clar:

Shell Sort	O(n)	O((nlog(n))^2)	O((nlog(n))^2)	O(1)



Iar comparând graficele desenate de mine pot presupune că qsort utilizează heapsort fiind că nu am observat ca programul să utilizeze memorie suplimentară în timpul efectuării qsort:

Heapsort	O(n log(n))	O(n log(n))	O(n log(n))	O(1)