Universidad del Caribe
2000 CANCUN, QUINTANA ROO, MÉXICO
CONOCIMIENTO Y CULTURA PARA EL DESARROLLO HUMANO

Técnicas Algoritmicas - IT0107

Tarea 01 del Primer Parcial
Profesor: **Dr. David Israel Flores Granados**.
Fecha de publicación: 24 de Agosto de 2018.
Fecha de entrega: 29 de Agosto de 2018.

Complejidad

- Elaborar un programa que implemente el Insertion Sort de manera ascendente sobre un conjunto de numeros.
- Elaborar un programa que implemente el Merge Sort de manera ascendente sobre un conjunto de números.
- Elaborar un programa que implemente el Bubble Sort de manera ascendente sobre un conjunto de numeros.
- Los tres programas deber an realizar lo siguiente:
  - Medir el tiempo de ejecución de cada algoritmo.
  - Leer el conjunto de numeros de los archivos: "n100.txt", "n1000.txt" y "n10000.txt"
- Crear los siguientes archivos:
  - "n100.txt" con 100 numeros enteros generados de manera aleatoria
  - "n1000.txt" con 1,000 numeros enteros generados de manera aleatoria
  - "n10000.txt" con 10,000 numeros enteros generados de manera aleatoria

Listing 1: Programa en C para calcular los tiempos en segundos de los algoritmos de métodos de ordenación.

```
/*
Descripcion: Implementar Insertion Sort, Merge Sort, Bubble Sort en C
    para leer y ordenar conjuntos de nmeros de archivos externos.
Programador: Joan de Jess Mndez Pool
Fecha de creacion: 24/08/2018
Entradas: 3 archivos externos (n100.txt, n1000.txt , n10000.txt)
Salida: Tabla comparativa del tiempo de ejecucin de los mtodos de
    ordenacin(Insertion, Merge, Bubble) para 100, 1000 y 10000 nmeros
    usando segundos como unidad de tiempo
*/
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

void control( double **, char **name);
void exe_time(double **, char **);
void create_files(char **name);
void interaction(int *, int);
void set_names(char **, char **);
void filerand(char *, int);
int countfile(char *);
int *readfile(char *, int);
double *set_timevar();
//Sorts
void insertion_sort(int *, int);
void merge_sort(int *, int, int);
void merge(int *, int, int, int);
void bubble_sort(int *, int);

void kill(int **);
void killdb(double **, int );

const int INF=1111;
int main(){
   int m=0;
   int *A=NULL;
   char **name=(char**)calloc(3, sizeof(char*)),
        **sort=(char**)calloc(3, sizeof(char*));
   double **total_time=NULL;
   time_t t;
   srand((unsigned) time(&t));

   total_time=set_timevar();
   set_names(name, sort);
   //create_files(name);

   A=readfile(name[0], 100);
```

```
44
45
46      control(total_time, name);
47      exe_time(total_time, sort);
48
49      killdb(total_time, 3);
50      printf("\n\n");
51      return 0;
52  }
53
54  void control(double **total_time, char **name){
55      clock_t start[3][3], end[3][3];
56      int *A=NULL;
57      int j=0, n=0;
58      for(int i=0; i<3; i++){
59          //Insertion Sort
60          n=countfile(name[i]);
61          A=readfile(name[i], n);
62          start[0][j] = clock();
63
64          insertion_sort(A, n);
65
66          end[0][j] = clock();
67          total_time[0][j] = ((double) (end[0][j] - start[0][j])) /
                CLOCKS_PER_SEC;//calulate total time
68          kill(&A);
69
70          //Merge Sort
71          n=countfile(name[i]);
72          A=readfile(name[i], n);
73          start[1][j] = clock();
74
75          merge_sort(A, 1, n);
76
77          end[1][j] = clock();
78          total_time[1][j] = ((double) (end[1][j] - start[1][j])) /
                CLOCKS_PER_SEC;//calulate total time
79          kill(&A);
80
81          //Bubble Sort
82          n=countfile(name[i]);
83          A=readfile(name[i], n);
84          start[2][j] = clock();
85
86          bubble_sort(A, n);
87
88          end[2][j] = clock();
89          total_time[2][j] = ((double) (end[2][j] - start[2][j])) /
                CLOCKS_PER_SEC;//calulate total time
90          j++;
```

```
      kill(&A);
   }
}
void exe_time(double **total_time, char **sort){
   printf("For\t\t100n\t\t1,000n\t\t10,000n\n\n");
   for(int i=0; i<3; i++){
      printf("%s\t", sort[i]);
      for(int j=0; j<3; j++){
         if(j==2){
            printf("%f\n", total_time[i][j]);
         }else{
            printf("%f\t", total_time[i][j]);
         }
      }
   }
}
void create_files(char **name){
   int m=100;
   for(int i=0; i<3; i++){
      filerand(name[i], m);
      m*=10;
   }
}
void interaction(int *ptr, int N){
   if(ptr!=NULL){
      for(int i=0; i<N; i++){
         if(i==N-1){
            printf("%d", ptr[i]);
         }else{
            printf("%d, ", ptr[i]);
         }
      }
   }else{
      printf("\nEmpty!\n");
   }
}
double *set_timevar(){
   double **time=(double**)calloc(3, sizeof(double*));
   for(int i=0; i<3; i++){
      time[i]=(double*) calloc(3, sizeof(double));
   }
   return time;
}
void set_names(char **name, char **sort){
   name[0]="/home/jj/CSource/C/Pointers/n100.txt";
   name[1]="/home/jj/CSource/C/Pointers/n1000.txt";
   name[2]="/home/jj/CSource/C/Pointers/n10000.txt";
   sort[0]="Insertion Sort";
   sort[1]="Merge Sort";
   sort[2]="Bubble Sort";
```

```
}
void filerand(char *handle, int n){
   FILE *api=fopen(handle, "w");
   int cad=0;
   for(int i=0; i<n; i++){
      cad=rand()%100+1;
      if(i==n-1){
         fprintf(api,"%d\0", cad);
      }else{
         fprintf(api,"%d\n", cad);
      }
   }
   fclose(api);
}
int countfile(char *handle){
   FILE *api=fopen(handle, "r");
   int id=0;
   char *p1;

   if(api != NULL){
      while ((p1 = fgetc(api)) != EOF){
         if(p1=='\n'){
            id++;
         }
      }
      fclose(api);
   }
   return ++id;
}
int *readfile(char *handle, int n){
   char line[10];
   int *ptr=(int*)calloc(n, sizeof(int));
   int id=0;
   FILE *api= fopen(handle, "r");

   while(fgets(line, 10, (FILE*) api)) {
      ptr[id++]=atoi(line);
   }
   fclose(api);
   return ptr;
}
//Sorts
void insertion_sort(int *A, int n){
   int key=0, i=0;
   for(int j=1; j<n; j++){
      key=A[j];
      i=j-1;
      while((i>=0)&&(key<A[i])){
         A[i+1]=A[i];
         i--;
```

```
191         }
192         A[i+1]=key;
193      }
194   }
195   void merge_sort(int *A, int p, int r){
196      int q=0;
197      if(p<r){
198         q=((p+r)/2);
199         merge_sort(A, p, q);
200         merge_sort(A, q+1, r);
201         merge(A, p, q, r);
202      }
203   }
204   void merge(int *A, int p, int q, int r){
205      int n1=q-p+1, n2=r-q;
206      int *L=(int*)calloc(n1+1, sizeof(int)), *R=(int*)calloc(n2+1,
           sizeof(int));
207      int i=0, j=0;
208
209      for(i=1; i<=n1; i++){
210         L[i-1]=A[p+i-2];
211      }
212      for(j=1; j<=n2; j++){
213         R[j-1]=A[q+j-1];
214      }
215      L[n1]=INF, R[n2]=INF;
216      i=0, j=0;
217      for(int k=p-1; k<r; k++){
218         if(L[i]<=R[j]){
219            A[k]=L[i++];
220         }else{
221            A[k]=R[j++];
222         }
223      }
224      kill(&L);
225      kill(&R);
226   }
227   void bubble_sort(int *A, int n ){
228      int key=0;
229      for(int i=0; i<n; i++){
230         for(int j=n-1; j>=i+1; j--){
231            if(A[j]<A[j-1]){
232               key=A[j];
233               A[j]=A[j-1];
234               A[j-1]=key;
235            }
236         }
237      }
238   }
239   void kill(int **ptr){
```

```
240     if(ptr[0]!=NULL){
241        free(*ptr);
242        *ptr=NULL;
243     }
244  }
245  void killdb(double **matriz, int N){
246     if(matriz[0]!=NULL){
247        for(int i=0; i<N; i++){
248           free(matriz[i]);
249           matriz[i]=NULL;
250        }
251        free(matriz);
252        matriz=NULL;
253     }
254  }
```

Output:

```
For             100n          1,000n          10,000n

Insertion Sort  0.000048      0.004504        0.268735
Merge Sort      0.000083      0.000606        0.005870
Bubble Sort     0.000122      0.008734        0.737409


jj@pcerdo:~$
```