TAREA PROGRAMADA I

REPORTE

LUIS EDUARDO ROJAS CARRILLO – B86875

I INTRODUCCIÓN:

En este trabajo se desarrollarán algoritmos utilizados para el ordenamiento de listas o vectores. Esto para poder realizar un estudio sobre el tiempo, funcionamiento y eficacia de dichos algoritmos.

Para este trabajo se plantean seis algoritmos distintos, los cuales son: MERGESORT, ORDENAMIENTO POR INSERCIÓN, ORDENAMIENTO POR SELECCIÓN, HEAPSORT, QUICKSORT y RADIXSORT.

El objetivo para esta tarea es ejecutar dichos algoritmos y compararlos, estudiando la eficacia según su tiempo de ejecución principalmente.

II METODOLOGÍA:

Para cumplir el objetivo se realizó el diseño y ejecución de los seis algoritmos planteados anteriormente para estudiar su funcionamiento. Posteriormente los algoritmos se pusieron a prueba ejecutando listas aleatoriamente desordenadas para así poder conocer los tiempos de ejecución de cada algoritmo en su promedio.

Para este trabajo se ejecutaron pruebas con listas de tamaños de: cincuenta mil, cien mil, ciento cincuenta mil y doscientos mil elementos desordenados aleatoriamente.

A continuación se plantean los cuadros de los promedios según el algoritmo y cantidad de datos.

Cuadro I

TIEMPO DE EJECUCION DE MERGE SORT

	50 000	100 000	150 000	200 000
1	0.010 sec	0.018 sec	0.031 sec	0.042 sec
2	0.010 sec	0.020 sec	0.027 sec	0.035 sec
3	0.010 sec	0.015 sec	0.025 sec	0.040 sec
PROMEDIO	0.010 sec	0.018 sec	0.028 sec	0.039 sec

En el cuadro I se observan los tiempos de ejecución del algoritmo MERGESORT con sus respectivos promedios.

Cuadro II

TIEMPO DE EJECUCION DE SELECCIÓN

	50 000	100 000	150 000	200 000
1	2.594 sec	10.454 sec	24.908 sec	42.007 sec
2	2.860 sec	11.063 sec	24.454 sec	41.815 sec
3	2.875 sec	11.000 sec	24.205 sec	41.925 sec
PROMEDIO	2.776 sec	10.839 sec	24.522 sec	41.915 sec

En el cuadro II se observan los tiempos de ejecución del algoritmo SELECCIÓN con sus respectivos promedios.

Cuadro III

TIEMPO DE EJECUCION DE INSERCIÓN

	50 000	100 000	150 000	200 000
1	1.699 sec	6.761 sec	14.176 sec	24.662 sec
2	1.585 sec	6.565 sec	14.706 sec	24.565 sec
3	1.779 sec	6.546 sec	14.290 sec	27.504 sec
PROMEDIO	1.687 sec	6.624 sec	14.391 sec	25.577 sec

En el cuadro III se observan los tiempos de ejecución del algoritmo INSERCIÓN con sus respectivos promedios.

Cuadro IV

TIEMPO DE EJECUCION DE HEAPSORT

	50 000	100 000	150 000	200 000
1	0.011 sec	0.023 sec	0.036 sec	0.048 sec
2	0.012 sec	0.024 sec	0.037 sec	0.044 sec
3	0.011 sec	0.024 sec	0.032 sec	0.040 sec
PROMEDIO	0.011 sec	0.024 sec	0.035 sec	0.044 sec

En el cuadro IV se observan los tiempos de ejecución del algoritmo HEAPSORT con sus respectivos promedios.

Cuadro V

TIEMPO DE EJECUCION DE QUICKSORT

	50 000	100 000	150 000	200 000
1	0.008 sec	0.019 sec	0.040 sec	0.064 sec
2	0.008 sec	0.021 sec	0.047 sec	0.074 sec
3	0.009 sec	0.023 sec	0.044 sec	0.063 sec
PROMEDIO	0.008 sec	0.021 sec	0.044 sec	0.067 sec

En el cuadro V se observan los tiempos de ejecución del algoritmo QUICKSORT con sus respectivos promedios.

Cuadro VI

TIEMPO DE EJECUCION DE RADIXSORT

	50 000	100 000	150 000	200 000
1	0.005 sec	0.008 sec	0.010 sec	0.014 sec
2	0.004 sec	0.007 sec	0.012 sec	0.018 sec
3	0.004 sec	0.009 sec	0.011 sec	0.015 sec
PROMEDIO	0.004 sec	0.007 sec	0.011 sec	0.015 sec

En el cuadro VI se observan los tiempos de ejecución del algoritmo RADIXSORT con sus respectivos promedios.

III RESULTADOS:

Como se ve en los seis cuadros anteriores con respecto a los promedios podemos ver una clara ventaja del algoritmo RADIXSORT sobre los cinco faltantes. A continuación se plantean los gráficos de cada uno de los algoritmos, según la cantidad de datos procesados y su tiempo de ejecución en segundos.

Gráfico I

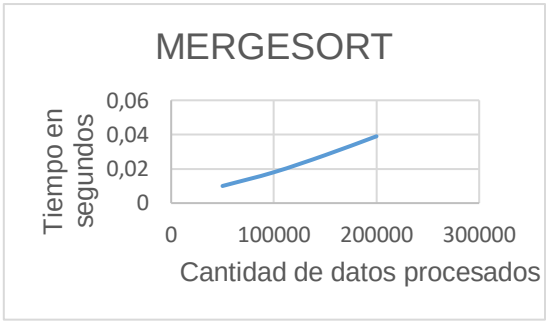


Gráfico II

Gráfico III

Gráfico IV

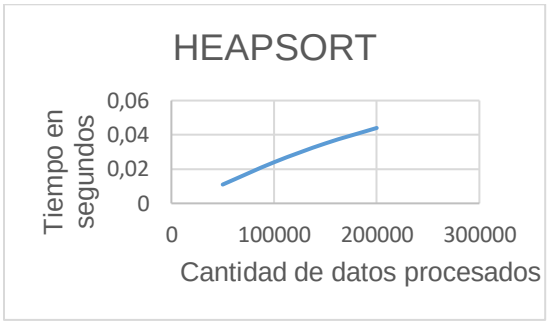


Gráfico V

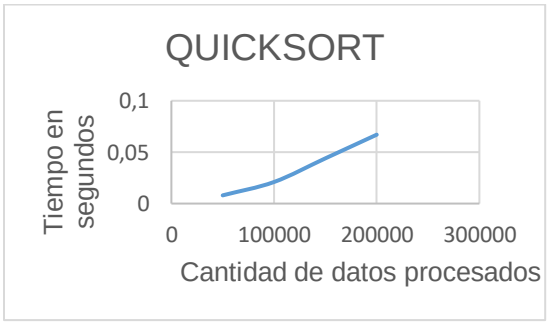


Gráfico VI

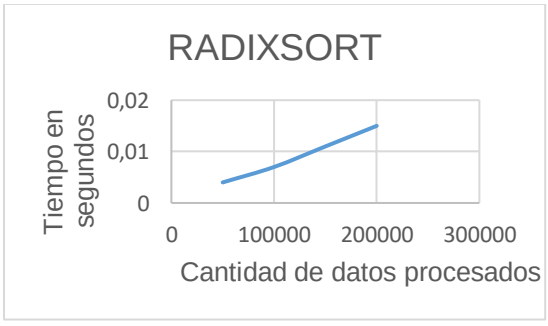
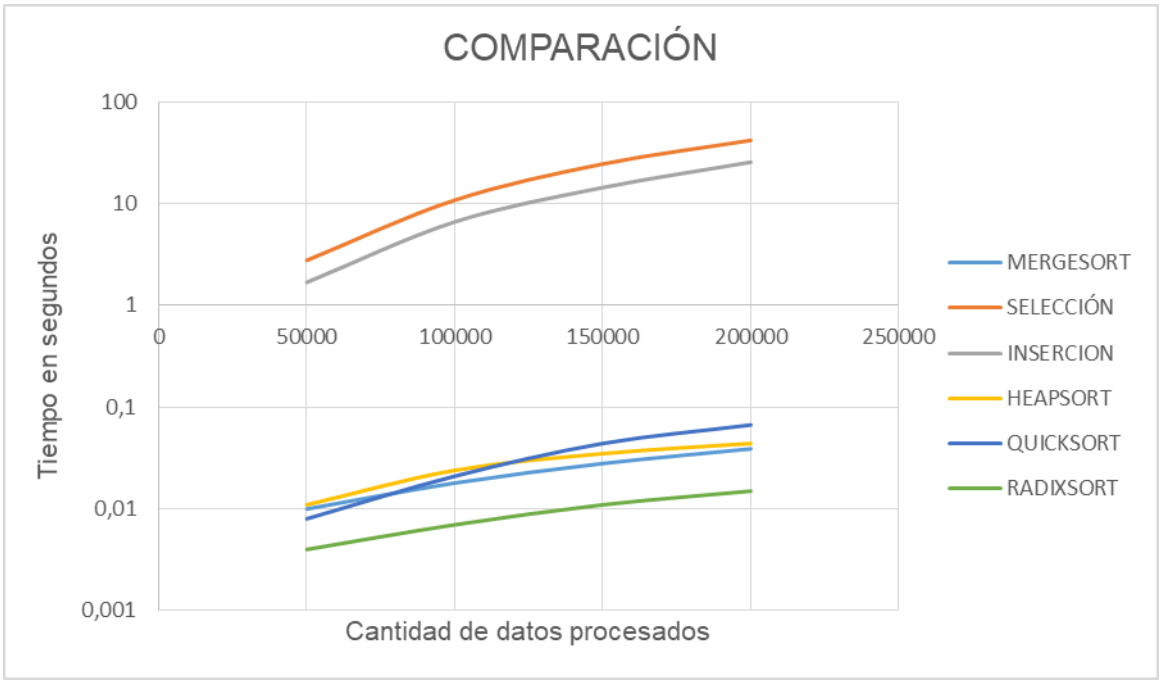


Gráfico VII



IV Conclusiones:

Como se puede notar en las gráficas y en las tablas antes propuestas, según el tiempo de ejecución de los seis algoritmos, el mejor algoritmo es el RADIXSORT.

Además podemos ver que los algoritmos de SELECCIÓN e INSERCIÓN se comportan de una manera relativamente parecida, siendo estos dos los peores algoritmos según su tiempo de ejecución.

Con este trabajo cumplimos el objetivo de ejecutar dichos algoritmos y ponerlos en práctica para conocer el mejor algoritmo según su tiempo de ejecución que como ya mencionamos, el mejor es el RADIXSORT y con una amplia ventaja, pero es bueno mencionar que el MERGESORT, HEAPSORT Y QUICKSORT se acerca bastante a su tiempo en promedio.

Algoritmo I

SELECCIÓN

Encargado de ordenar la lista mediante el algoritmo de selección.

```
void Ordenador::seleccion(int *arreglo, int tamano){
  for(int i=0; i<tamano-1;i++){
      int menor=i;
      for(int j=i+1; j<tamano; j++){
          if(arreglo[j]<arreglo[menor]){
              menor=j;
          }
        }
      int temp=arreglo[i];
      arreglo[i]=arreglo[menor];
      arreglo[menor]=temp;
    }
}
```

Algoritmo II

INSERCIÓN

Encargado de ordenar la lista mediante el algoritmo de inserción.

```
void Ordenador::insercion(int *arreglo, int
    for(int j=1; j<tamano; j++){
        int llave=arreglo[j];
        int i=j-1;
        while(i>=0 && arreglo[i]>llave){
            arreglo[i+1]=arreglo[i];
            i=i-1;
        }
        arreglo[i+1]=llave;
    }
}
```

Algoritmo III

MERGESORT

Encargado de ordenar la lista mediante el algoritmo de mergesort.

```
void Ordenador::MergeSort(int *arreglo, i
    if(p<r){
        int q=((p+r)/2);
        MergeSort(arreglo,p,q);
        MergeSort(arreglo,q+1,r);
        Merge(arreglo,p,q,r);
    }
}
```

```
void Ordenador::Merge(int *arreglo, int p, int
    int n1=q-p+1;
    int n2=r-q;
    int izq[n1+1];
    int der[n2+1];
    izq[n1]=INFINITO;
    der[n2]=INFINITO;
    for(int i=0; i<n1; i++){
        izq[i]=arreglo[p+i];
    }
    for(int i=0; i<n2; i++){
        der[i]=arreglo[q+i+1];
    }
    int i=0;
    int j=0;
    for(int k=p;k<=r;k++){

        if(izq[i]<der[j]){
            arreglo[k]=izq[i];
            i++;
        }else{
            arreglo[k]=der[j];
            j++;
        }
    }
}
```

Algoritmo IV

HEAPSORT

Encargado de ordenar la lista mediante el algoritmo de heapsort.

```
void Ordenador::heapsort(int *arreglo, int tamano){
    monticularizar(arreglo,tamano-1);
    int i=tamano-1;
    while(i>=0){
        arreglo[i]=extraer_max(arreglo,i);
        i--;
    }

}
```

```
void Ordenador::corregir_cima(int *arreglo, int k, int n){
    int j;
    if(arreglo[2*k]>arreglo[2*k+1]){
        j=2*k;
    }else{
        j=2*k+1;
    }

    if(arreglo[j]>arreglo[k]){
        int temp=arreglo[k];
        arreglo[k]=arreglo[j];
        arreglo[j]=temp;
        if(j*2+1<=n){
        corregir_cima(arreglo,j,n);
        }
    }
}

void Ordenador::monticularizar(int *arreglo, int n){
    for(int i=(n/2);i>=0;i--){
        corregir_cima(arreglo,i,n);
    }
}
```

Algoritmo V

QUICKSORT

Encargado de ordenar la lista mediante el algoritmo de quicksort.

```
void Ordenador::Quicksort(int * arreglo, int p
    if (p<r) {
        int q = partition(arreglo, p, r);
        Quicksort(arreglo,p, q-1);
        Quicksort(arreglo, q+1, r);
    }

}

int Ordenador::partition(int * arreglo, int
    int x = arreglo[r];
    int i = p-1;
    for (int j=p; j<=r-1; j++) {
        if (arreglo[j]<=x) {
            i++;
            int t1 = arreglo[i];
            arreglo[i] = arreglo[j];
            arreglo[j] = t1;
        }
    }
    int t2 = arreglo[i+1];
    arreglo[i+1] = arreglo[r];
    arreglo[r] = t2;
    return (i + 1);
}
```

Algoritmo VI

RADIXSORT

Encargado de ordenar la lista mediante el algoritmo de radixsort.

```
void Ordenador::radixsort(int* arreglo, int
    int valor= abs(obtenerMinimo(arreglo,tam
    for(int i=0;i<tamano;i++){
        arreglo[i]=arreglo[i]+valor;
    }
    int m = getMax(arreglo, tamano);
    for (int exp = 1; m/exp > 0; exp *= 10)
        countsort(arreglo, tamano, exp);
    }
    for(int i=0;i<tamano;i++){
        arreglo[i]=arreglo[i]-valor;
    }
}
```

```
void Ordenador::countsort(int *arreglo, int n, int exp){//COUNTSO
    int output[n];
    int i, count[10] = {0};
    for (i = 0; i < n; i++){
        count[ (arreglo[i]/exp)%10 ]++;
    }
    for (i = 1; i < 10; i++) {
        count[i] += count[i - 1];
    }
    for (i = n - 1; i >= 0; i--) {
        output[count[ (arreglo[i]/exp)%10 ] - 1] = arreglo[i];
        count[ (arreglo[i]/exp)%10 ]--;
    }
    for (i = 0; i < n; i++) {
        arreglo[i] = output[i];
    }
}
```

V Referencias:

-Cormen, T. H., Leiserson, C. E., Rivest, R. L.,, Stein, C. Introduction to Algorithms. The MIT Press, 2001.

-Hernández, Z.J. y otros: Fundamentos de Estructuras de Datos. Soluciones en Ada, Java y C++, Thomson, 2005.

-Weiss, M.A.: Data Structures and Algorithm Analysis in C++, 4th Edition,Pearson/Addison Wesley, 2014.