**Assignment No.2**

**Bubble Sort-**

#include <stdio.h>

#include <time.h>

int main(){

int array[1000], n, c, d,swap, sp=0,comp=0,pos;

printf("Enter Size\n");

scanf("%d", &n);

for (c = 0; c < n; c++) scanf("%d", &array[c]);

for (c=0;c<(n-1);c++){

for (d=0;d<n-c-1;d++){

comp++;

if (array[d] > array[d+1]){

swap = array[d]; array[d] = array[d+1]; array[d+1]= swap; sp++; }}}

printf("Bubble Sort:\n");

for (c=0;c<n;c++) printf("%d\n", array[c]);

printf("Comparision:%d\n",comp);

printf("Swapping:%d\n",sp);

}

**Algorithm: BubbleSort**

**Input:** Unsorted Array

**Output:** Sorted Array

1. begin BubbleSort(list)
2. for all elements of list
   1. if list[i] > list[i+1]
      1. swap(list[i], list[i+1])

2.2. end if

3. end for

4. return list

5. end BubbleSort

**Selection Sort-**

#include<stdio.h>

#include<stdlib.h>

#include <time.h>

int main(){

int i,n,s,temp,swap=0,comp=0;

scanf("%d",&n);

int A[n];

for(i=0;i<n;++i) scanf("%d",&A[i]);

for(s=0;s<n;++s)

for(i=s+1;i<n;++i)

{ comp++;

if(A[s]>A[i]){

temp=A[s]; A[s]=A[i]; A[i]=temp; swap++; }

}

for(i=0;i<n;++i) printf("%d ",A[i]);

printf("Total Swapping: %d\nTotal Comparison: %d\n",swap,comp);

return 0;

}

**Algorithm: SelectionSort**

**Input:** Unsorted Array

**Output:** Sorted Array

**Step 1** − Set MIN to location 0

**Step 2** − Search the minimum element in the list

**Step 3** − Swap with value at location MIN

**Step 4** − Increment MIN to point to next element

**Step 5** − Repeat until list is sorted

**Insertion Sort-**

#include <stdio.h>

#include <math.h>

#include <time.h>

int main(){

int n, c, d,swap, sp=0,comp=0,pos;

printf("Enter Size\n");

scanf("%d", &n);

int arr[n];

printf("Enter elements\n");

for (c = 0; c < n; c++) scanf("%d", &arr[c]);

int i, key, j;

for (i = 1; i < n; i++){ comp++; key = arr[i]; j = i-1;

while (j >= 0 && arr[j] > key) {

comp++; sp++;

arr[j+1] = arr[j];

j = j-1; }

arr[j+1] = key; }

for (i=0; i < n; i++) printf("%d ", arr[i]);

printf("Comparision:%d\n",comp);

printf("Swapping:%d\n",sp);

return 0;

}

**Algorithm: InsertionSort**

**Input:** Unsorted Array

**Output:** Sorted Array

**Step 1** − If it is the first element, it is already sorted. return 1;

**Step 2** − Pick next element

**Step 3** − Compare with all elements in the sorted sub-list

**Step 4** − Shift all the elements in the sorted sub-list that is greater than the

value to be sorted

**Step 5** − Insert the value

**Step 6** − Repeat until list is sorted

**Quick Sort-**

#include<stdio.h>

int comp=0,sp=0;

void swap(int\* a, int\* b){

int t = \*a; \*a = \*b; \*b = t; }

int partition (int arr[], int low, int high){

int pivot = arr[high], i = (low - 1),j;

for (j = low; j <= high- 1; j++){

if (arr[j] <= pivot){ comp++; i++; swap(&arr[i], &arr[j]); } }

swap(&arr[i + 1], &arr[high]); sp++;

return (i + 1); }

void quickSort(int arr[], int low, int high){

if (low < high){

int pi = partition(arr, low, high); quickSort(arr, low, pi - 1); quickSort(arr, pi + 1, high); } }

int main(){

int n,i,j; scanf("%d",&n); int arr[n];

printf("Enter Original array\n");

for(i=0;i<n;i++) scanf("%d",&arr[i]);

quickSort(arr, 0, n-1);

for(i=0;i<n;i++) printf("%d ",arr[i]);

printf("Swapping:%d\n",sp); }

**Algorithm: QuickSort**

**Input:** Unsorted Array

**Output:** Sorted Array

**Step 1** − Choose the highest index value has pivot

**Step 2** − Take two variables to point left and right of the list excluding pivot

**Step 3** − left points to the low index

**Step 4** − right points to the high

**Step 5** − while value at left is less than pivot move right

**Step 6** − while value at right is greater than pivot move left

**Step 7** − if both step 5 and step 6 does not match swap left and right

**Step 8** − if left ≥ right, the point where they met is new pivot

**Merge Sort-**

#include <stdio.h>

#include <time.h>

void MERGEALGO(int [], int, int, int); void DIVIDE(int [],int, int); int comp=0,sp=0;

int main(){

int ARRAY[1000]; int i, LENGTH;

printf("Enter total number of elements:");

scanf("%d", &LENGTH);

printf("Enter the elements:\n");

for(i = 0; i < LENGTH; i++) scanf("%d", &ARRAY[i]);}

clock\_t begin = clock();

DIVIDE(ARRAY, 0, LENGTH - 1);

printf("After merge sort:\n");

for(i = 0;i < LENGTH; i++) printf("%d ",ARRAY[i]);

clock\_t end = clock();

double time\_spent = (double)(end - begin) / CLOCKS\_PER\_SEC;

printf("----------------%lf------------------",time\_spent); }

void DIVIDE(int ARRAY[],int low,int high){

int mid;

if(low < high){

comp++; mid = (low + high) / 2;

DIVIDE(ARRAY, low, mid); DIVIDE(ARRAY, mid + 1, high); MERGEALGO(ARRAY, low, mid, high); } }

void MERGEALGO(int ARRAY[],int low,int mid,int high){

int i, mi, k, lo, temp[1000];

lo = low;

i = low;

mi = mid + 1;

while ((lo <= mid) && (mi <= high)){

if (ARRAY[lo] <= ARRAY[mi]){

comp++; temp[i] = ARRAY[lo]; lo++; }

else{ temp[i] = ARRAY[mi]; mi++; }

i++;}

if (lo > mid){ comp++;

for (k = mi; k <= high; k++){

temp[i] = ARRAY[k]; i++; sp++; }}

else{

for (k = lo; k <= mid; k++){

temp[i] = ARRAY[k]; i++; sp++; }}

for (k = low; k <= high; k++) ARRAY[k] = temp[k];

}

**Algorithm: MergeSort**

**Input:** Unsorted Array

**Output:** Sorted Array

**Step 1** − if it is only one element in the list it is already sorted, return.

**Step 2** − divide the list recursively into two halves until it can no more be divided.

**Step 3** − merge the smaller lists into new list in sorted order.