Name: Parth Savla Roll No : D135 Div:D2/2

EXPERIMENT NO. 1 SELECTION AND QUICK SORT

**AIM**: Implementation of Selection and Quick Sort.

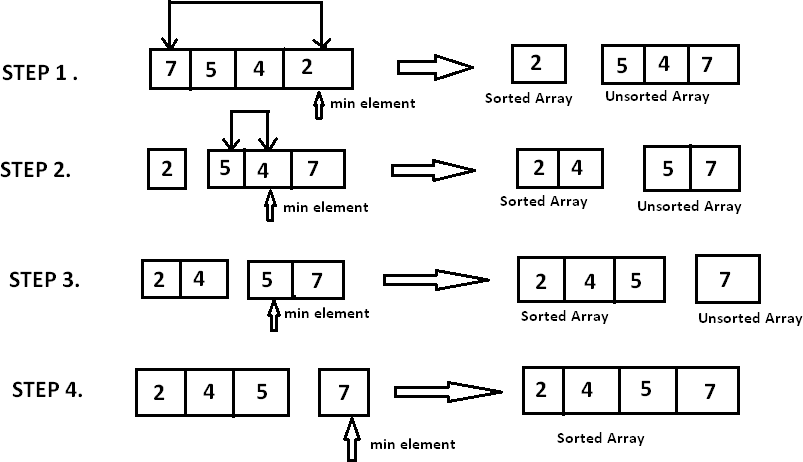
**THEORY:**

# Selection Sort

Selection Sort is a comparison-based sorting algorithm. It sorts an array by repeatedly selecting the smallest (or largest) element from the unsorted portion and swapping it with the first unsorted element. This process continues until the entire array is sorted.

1. First we find the smallest element and swap it with the first element. This way we get the smallest element at its correct position.
2. Then we find the smallest among remaining elements (or second smallest) and swap it with the second element.
3. We keep doing this until we get all elements moved to correct position.

Here’s a basic overview of how the Selection Sort works.



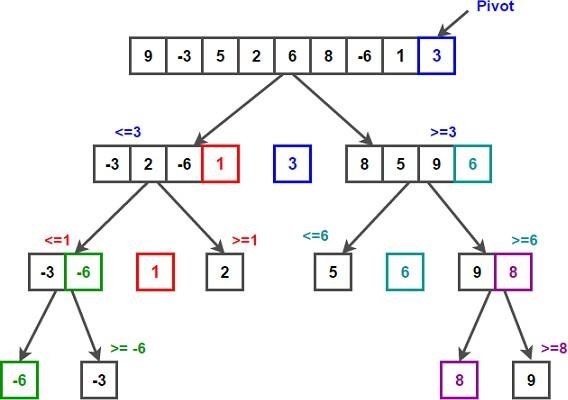
# Quick Sort

Quick Sort works on the principle of divide and conquer, breaking down the problem into smaller sub-problems.

There are mainly three steps in the algorithm:

1. Choose a Pivot: Select an element from the array as the pivot. The choice of pivot can vary (e.g., first element, last element, random element, or median).
2. Partition the Array: Rearrange the array around the pivot. After partitioning, all elements smaller than the pivot will be on its left, and all elements greater than the pivot will be on its right. The pivot is then in its correct position, and we obtain the index of the pivot.
3. Recursively Call: Recursively apply the same process to the two partitioned sub- arrays (left and right of the pivot).
4. Base Case: The recursion stops when there is only one element left in the sub- array, as a single element is already sorted.

Here’s a basic overview of how the Quick Sort algorithm works.



**ALGORITHM:**

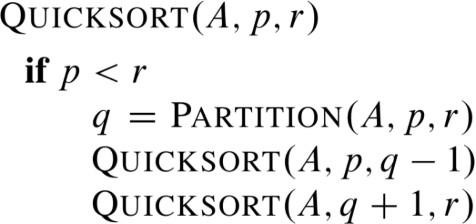
**Selection Sort**

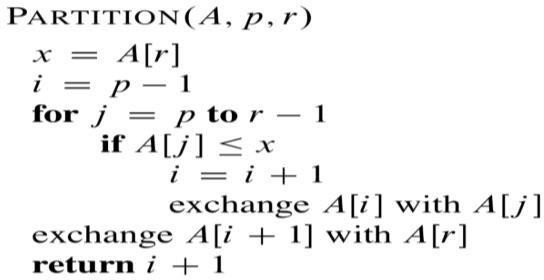
1. Start.
2. Define an array ‘A’ of size ‘n’.
3. Read array elements from the user.
4. Define first element of the array as ‘min’ i.e. min=A[0].
5. For i0 to n-1
   1. minA[i]
   2. loci //loc variable keeps the track of minimum element
   3. for ji+1 to n-1
      1. if (A[j]<min) then
         1. minA[j]
         2. locj
   4. if (loc!=i) then
   5. swap( A[i] , A[loc])
6. Display sorted array.
7. Stop.

# Quick Sort

In Quick sort algorithm, partitioning of the list is performed using following steps:

1. Consider the first element of the list as pivot (i.e., Element at first position in the list).
2. Define two variables i and j. Set i and j to first and last elements of the list respectively.
3. Increment i until list[i] > pivot then stop.
4. Decrement j until list[j] < pivot then stop.
5. If i < j then exchange list[i] and list[j].
6. Repeat steps 3,4 & 5 until i > j.
7. Exchange the pivot element with list[j] element.





**SELECTION SORT:**

**CODE**:

#include<stdio.h>

#include<conio.h>

int main() {

int arr[8];

int i, j, minIndex, temp;

clrscr();

printf("Enter the elements in array : ");

for(i=0;i<6;i++)

{

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

}

printf("Original array: \n");

for (i = 0; i < 6; i++) {

printf("%d ", arr[i]);

}

printf("\n");

for(i=0;i<6-1;i++)

{

minIndex = i;

for(j=i+1;j<6;j++)

{

if(arr[j]<arr[minIndex])

{

minIndex = j;

}

}

if(i!=minIndex)

{

temp = arr[i];

arr[i] = arr[minIndex];

arr[minIndex]=temp;

}

}

printf("Sorted array: \n");

for (i = 0; i < 6; i++) {

printf("%d ", arr[i]);

}

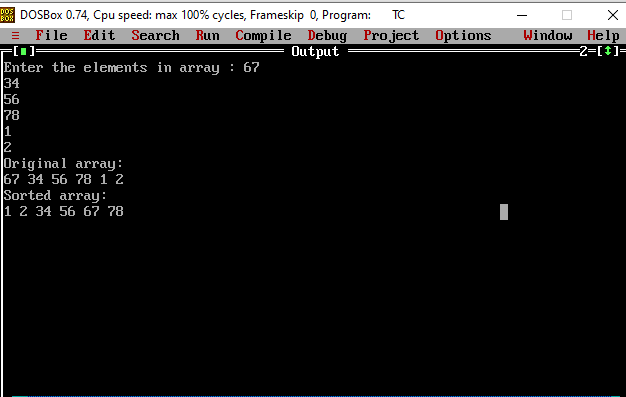
printf("\n");

getch();

return 0;

}

**OUTPUT:**



**QUICK SORT:**

**CODE:**

#include <stdio.h>

#include <conio.h>

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

int pivot = arr[high];

int i = low - 1;

int j, temp;

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

if (arr[j] <= pivot) {

i++;

temp = arr[i];

arr[i] = arr[j];

arr[j] = temp;

}

}

temp = arr[i + 1];

arr[i + 1] = arr[high];

arr[high] = temp;

return i + 1;

}

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

int pi;

if (low < high) {

pi = partition(arr, low, high);

quickSort(arr, low, pi - 1);

quickSort(arr, pi + 1, high);

}

}

void printArray(int arr[], int size) {

int i;

for (i = 0; i < size; i++) {

printf("%d ", arr[i]);

}

printf("\n");

}

int main() {

int arr[8];

int n, i;

clrscr();

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

scanf("%d", &n);

printf("Enter %d elements: \n", n);

for (i = 0; i < n; i++) {

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

}

printf("Original array: \n");

printArray(arr, n);

quickSort(arr, 0, n - 1);

printf("Sorted array: \n");

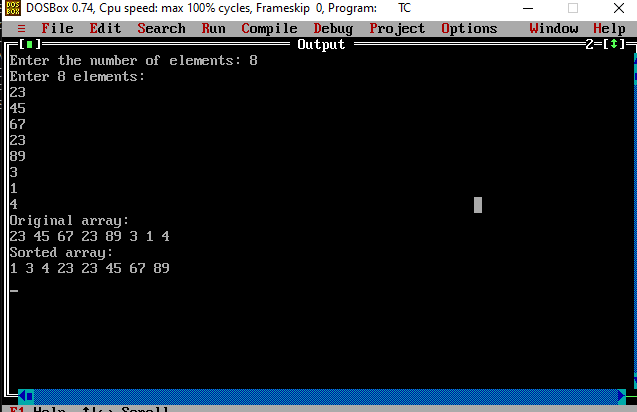
printArray(arr, n);

getch();

return 0;

}

**OUTPUT:**



**ANALYSIS OF TIME COMPLEXITY:**

**SELECTION SORT:** O(n2)

**QUICK SORT**: O( n log n )

**CONCLUSION**: Implemented and analyzed Selection and Quick sort successfully.