EXPERIMENT - 11

Data Structures

Aim

To implement Insertion sort and Selection sort techniques using array.

Syeda Reeha Quasar
14114902719
3C7

Name - Syeda Reeha Quasar **Roll no. – 14114802719** **Group – C7**

EXPERIMENT – 11

AIM: To implement Insertion sort and Selection sort techniques using array.

THEORY

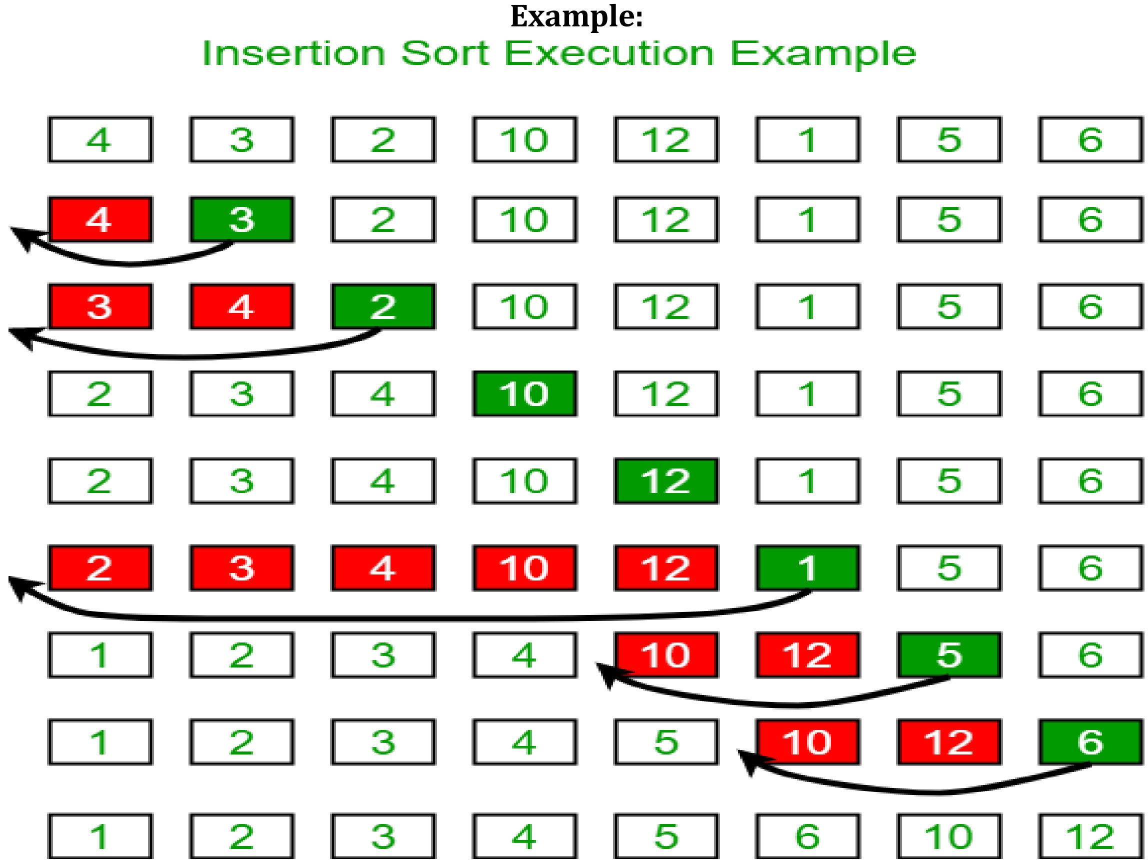
Insertion Sort

Insertion sort is a simple sorting algorithm that works similar to the way you sort playing cards in your hands. The array is virtually split into a sorted and an unsorted part. Values from the unsorted part are picked and placed at the correct position in the sorted part.

Algorithm

To sort an array of size n in ascending order:
1: Iterate from arr[1] to arr[n] over the array.
2: Compare the current element (key) to its predecessor.
3: If the key element is smaller than its predecessor, compare it to the elements before. Move the greater elements one position up to make space for the swapped element.

Example:

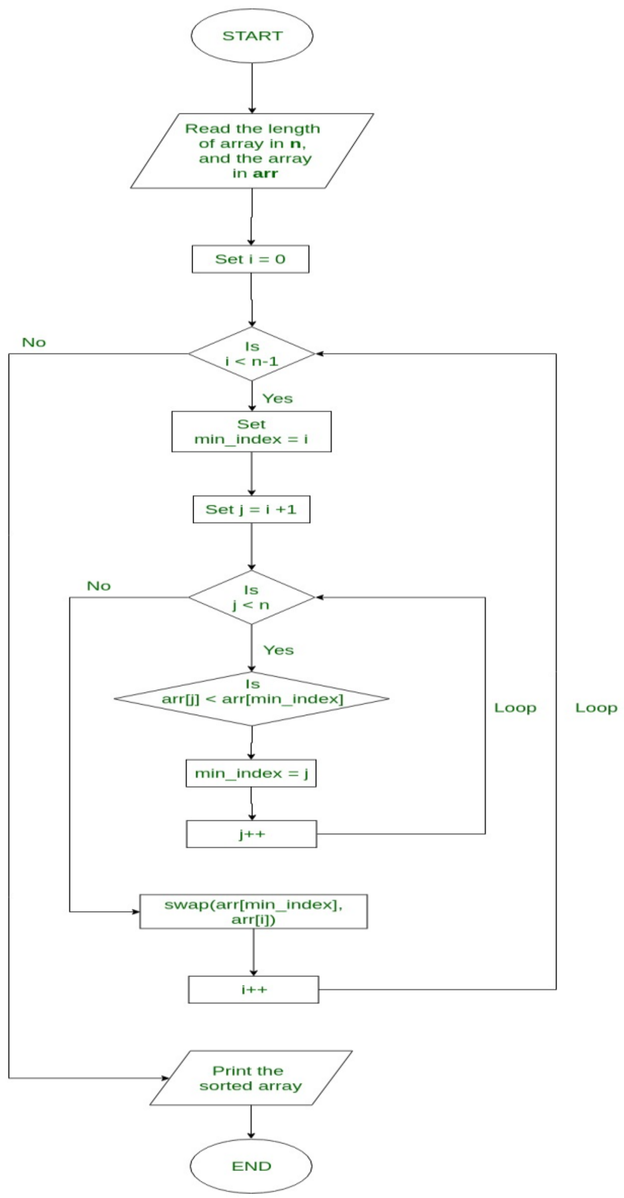


Selection Sort

The selection sort algorithm sorts an array by repeatedly finding the minimum element (considering ascending order) from unsorted part and putting it at the beginning. The algorithm maintains two subarrays in a given array.

1) The subarray which is already sorted.
2) Remaining subarray which is unsorted.

In every iteration of selection sort, the minimum element (considering ascending order) from the unsorted subarray is picked and moved to the sorted subarray.



Flowchart for Selection Sort

arr[] = 64 25 12 22 11

// Find the minimum element in arr[0...4]

// and place it at beginning

11 25 12 22 64

// Find the minimum element in arr[1...4]

// and place it at beginning of arr[1...4]

11 **12** 25 22 64

// Find the minimum element in arr[2...4]

// and place it at beginning of arr[2...4]

11 12 **22** 25 64

// Find the minimum element in arr[3...4]

// and place it at beginning of arr[3...4]

11 12 22 **25** 64

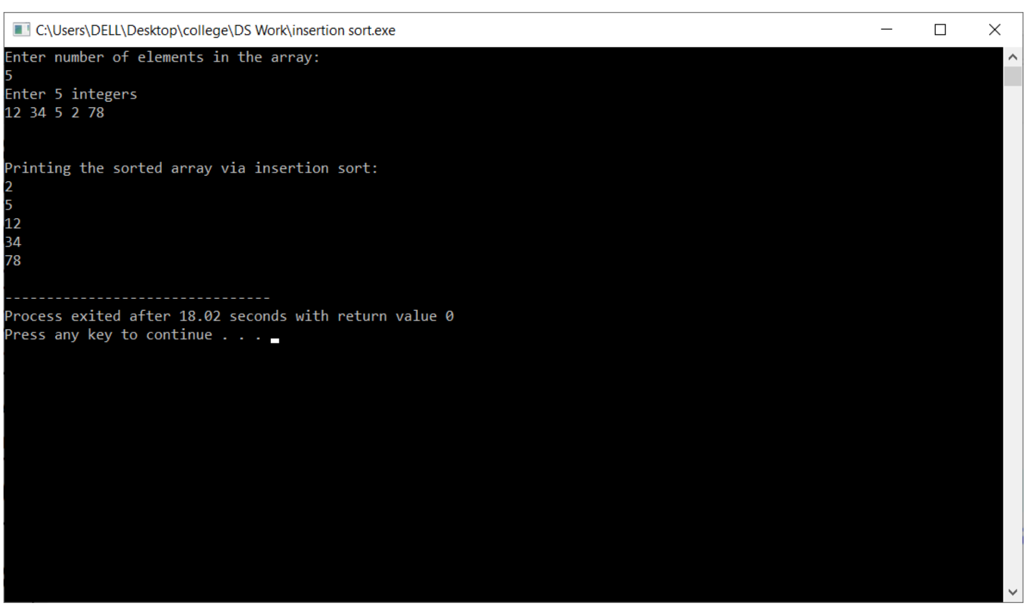
Insertion Sort

Source code:

```
#include <math.h>

#include <stdio.h>


// Function to sort an array using insertion sort

void insertionSort(int arr[], int n){

   int i, key, j; // variables


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

      key = arr[i];

      j = i - 1;


      // Move elements of arr[0..i-1], that are greater than key, to one position ahead of their current
position

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

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

         j = j - 1;

      }


      arr[j + 1] = key;

   }

}


int main(){

  int arr[100], n, i; // arr and variable declaration


 // taking array size

  printf("Enter number of elements in the array:\n");
```

```
 scanf("%d", &n);


 printf("Enter %d integers\n", n);


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


 insertionSort(arr, n); // insertion sorting


 printf("\n\nPrinting the sorted array via insertion sort:\n");


// printing sorted array
 for (i = 0; i < n; i++)
    printf("%d\n", arr[i]);


 return 0;
}
```

OUTPUT



Selection Sort

Source code:

```
#include <stdio.h> // req lib

// function for swapping
void swap(int *xp, int *yp){ // swapping x and y pointers using 3 variable approach
   int temp = *xp;
   *xp = *yp;
   *yp = temp;
}

void selectionSort(int arr[], int n) {
   int i, j, min_idx;

   // One by one move boundary of unsorted subarray
   for (i = 0; i < n-1; i++) {
      min_idx = i; // Finding the minimum element in unsorted array
      for (j = i+1; j < n; j++) {
       if (arr[j] < arr[min_idx])
         min_idx = j;
         }

      swap(&arr[min_idx], &arr[i]);   // Swap the found minimum element with the first element
   }
}

// Driver program to test above functions
int main()
```

```
{
    int arr[100], n, i; // arr and variable declaration

  // taking array size
   printf("Enter number of elements in the array:\n");
   scanf("%d", &n);

   printf("Enter %d integers\n", n);

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

   selectionSort(arr, n); // selection sorting

   printf("\n\nPrinting the sorted array:\n");

  // printing sorted array
   for (i = 0; i < n; i++)
      printf("%d\n", arr[i]);

   return 0;
}
```

OUTPUT

```
C:\Users\DELL\Desktop\college\DS Work\selection Sort.exe
Enter number of elements in the array:
5
Enter 5 integers
23 4 32 6 1


Printing the sorted array:
1
4
6
23
32

--------------------------------
Process exited after 17.59 seconds with return value 0
Press any key to continue . . .
```

VIVA VOICE

Q1. ***What is an in-place sorting algorithm?***

a) It needs O(1) or O(logn) memory to create auxiliary locations
b) The input is already sorted and in-place
c) It requires additional storage
d) It requires additional space

Ans.

Answer: a
Explanation: Auxiliary memory is required for storing the data temporarily.

Q2. ***In the following scenarios, when will you use selection sort?***

a) The input is already sorted
b) A large file has to be sorted
c) Large values need to be sorted with small keys
d) Small values need to be sorted with large keys

Ans.

Answer: c
Explanation: Selection is based on keys, hence a file with large values and small keys can be efficiently sorted with selection sort.

Q3. What is the worst case complexity of selection sort?
a) O(nlogn)
b) O(logn)
c) O(n)
d) $O(n^2)$

Ans.

Answer: d
Explanation: Selection sort creates a sub-list, LHS of the 'min' element is already sorted and RHS is yet to be sorted. Starting with the first element the 'min' element moves towards the final element.

Q4. What is the advantage of selection sort over other sorting techniques?
a) It requires no additional storage space
b) It is scalable
c) It works best for inputs which are already sorted
d) It is faster than any other sorting technique

Ans.

Answer: a
Explanation: Since selection sort is an in-place sorting algorithm, it does not require additional storage.

Q5. What is the average case complexity of selection sort?
a) O(nlogn)
b) O(logn)
c) O(n)
d) $O(n^2)$

Ans.

Answer: d
Explanation: In the average case, even if the input is partially sorted, selection sort behaves as if the entire array is not sorted. Selection sort is insensitive to input.

Q6. Which of the following is correct with regard to insertion sort?
a) insertion sort is stable and it sorts In-place
b) insertion sort is unstable and it sorts In-place
c) insertion sort is stable and it does not sort In-place
d) insertion sort is unstable and it does not sort In-place

Ans.

Answer: a
Explanation: During insertion sort, the relative order of elements is not changed. Therefore, it is a stable sorting algorithm. And insertion sort requires only O(1) of additional memory space. Therefore, it sorts In-place.

Q7. Which of the following sorting algorithm is best suited if the elements are already sorted?
a) Heap Sort
b) Quick Sort
c) Insertion Sort
d) Merge Sort

Ans.

Answer: c
Explanation: The best case running time of the insertion sort is O(n). The best case occurs when the input array is already sorted. As the elements are already sorted, only one comparison is made on each pass, so that the time required is O(n).

Q8. The worst case time complexity of insertion sort is $O(n^2)$. What will be the worst case time complexity of insertion sort if the correct position for inserting element is calculated using binary search?
a) O(nlogn)
b) $O(n^2)$
c) O(n)
d) O(logn)

Ans.

Answer: b
Explanation: The use of binary search reduces the time of finding the correct position from O(n) to O(logn). But the worst case of insertion sort remains $O(n^2)$ because of the series of swapping operations required for each insertion.

Q9. Insertion sort is an example of an incremental algorithm.
a) True
b) False

Ans.

Answer: a
Explanation: In the incremental algorithms, the complicated structure on n items is built by first building it on n − 1 items. And then we make the necessary changes to fix things in adding the last item. Insertion sort builds the sorted sequence one element at a time. Therefore, it is an example of an incremental algorithm.

Q10. Which of the following is good for sorting arrays having less than 100 elements?

a) Quick Sort
b) Selection Sort
c) Merge Sort
d) Insertion Sort

Ans.

Answer: d
Explanation: The insertion sort is good for sorting small arrays. It sorts smaller arrays faster than any other sorting algorithm.