**Experiment No. : 1**

**Title: Basic Sorting algorithm and its analysis**

**Batch:B2 Roll No.:160104211191 Experiment No.: 1**

**Aim:** To implement and analyse time complexity of insertion sort & Heap sort.

**Explanation and Working of insertion sort & Heap sort:**

1. **Insertion Sort----**

**Explaination:- Insertion sort is a simple sorting algorithm that works by iteratively building a sublist one item at a time. The algorithm works by comparing each item in the unsorted portion of the list to the items in the sorted portion of the list, and inserting the current item into its correct position in the sorted sublist.**

**Working: - Here are the steps involved in performing an insertion sort:**

1. **Start with the first element in the list. This is considered the sorted sublist of length**
2. **Take the next element in the unsorted portion of the list, and compare it to the items in the sorted sublist from right to left. If the current element is smaller than an element in the sorted sublist, shift that element to the right by one position.**
3. **Repeat step 2 until the current element is no longer smaller than the element to its left in the sorted sublist.**
4. **Insert the current element into its correct position in the sorted sublist.**
5. **Repeat steps 2-4 for all remaining elements in the unsorted portion of the list.**
6. **When all elements have been processed, the list is sorted.**
7. **Heap Sort----**

**Explaination: - Heap sort is a sorting algorithm that uses the heap data structure to sort a list of elements. A heap is a binary tree in which each parent node is greater than or equal to its children (in a max heap) or less than or equal to its children (in a min heap). Heap sort works by first building a max heap from the input list, then repeatedly extracting the maximum element from the heap and placing it at the end of the sorted portion of the list.**

**Working:- Here are the steps involved in performing a heap sort:**

1. **Build a max heap from the input list. This is done by repeatedly swapping elements in the list until the binary tree satisfies the heap property.**
2. **Extract the maximum element from the heap and place it at the end of the sorted portion of the list.**
3. **Restore the heap property by swapping elements in the heap as necessary.**
4. **Repeat steps 2-3 for all remaining elements in the heap.**
5. **When all elements have been extracted, the list is sorted in descending order. To obtain a sorted list in ascending order, reverse the order of the elements.**

**Algorithm of insertion sort & Heap sort:**

1. **Insertion Sort**

**Algorithm:-**

1. **Start with the second element in the list, since a list of one element is already sorted.**
2. **Compare the second element with the first element. If the second element is smaller, swap the two elements.**
3. **Move on to the third element, and compare it with the second element. If the third element is smaller, swap it with the second element. Then, compare the new second element with the first element, and swap if necessary.**
4. **Continue this process for all remaining elements in the list, comparing each element with the previous elements and swapping as necessary.**
5. **When all elements have been processed, the list is sorted.**
6. **Heap Sort**

**Algorithm:-**

1. **Build a max heap from the input list.**
2. **Extract the maximum element from the heap and place it at the end of the sorted portion of the list.**
3. **Restore the heap property by swapping elements in the heap as necessary.**
4. **Repeat steps 2-3 for all remaining elements in the heap.**
5. **When all elements have been extracted, the list is sorted in descending order. To obtain a sorted list in ascending order, reverse the order of the elements.**

**Derivation of Analysis** **insertion sort & Heap sort:**

Worst Case Analysis:-

In the worst case, the inner while loop of the insertion sort algorithm will compare the current element with every element to its left in the sorted sublist before finding its correct position. This means that for the i-th element, the while loop will execute i-1 times.

Therefore, the total number of comparisons in the worst case is:

1 + 2 + ... + (n-1) = n(n-1)/2

Since each comparison takes constant time, the time complexity of insertion sort in the worst case is O(n^2).

Best Case Analysis

In the best case, the input list is already sorted, so the while loop in the algorithm will never execute, and the time complexity is O(n).

Average Case Analysis

In the average case, the number of comparisons is roughly half of the worst case, which gives an average time complexity of O(n^2).

**Program(s) of insertion sort & Heap sort:**

1. Insertion Sort..

#include <iostream>

#include <cstdlib>

using namespace std;

void insertionSort(int arr[], int size)

{

int key;

int temp;

int shiftcount = 0;

int sum = 0;

for (int i = 1; i < size; i++)

{

key = arr[i];

int j = i - 1;

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

{

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

j = j - 1;

shiftcount = shiftcount + 1;

}

cout << "\nThe number of shifts in cycle " << i << " is: " << shiftcount;

temp = shiftcount;

sum = sum + temp;

cout << "\n";

shiftcount = 0;

arr[j + 1] = key;

}

cout << "\n";

cout << "Total shifts are -- " << sum;

}

void printarr(int arr[], int n)

{

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

{

cout << arr[i] << " ";

}

}

int main()

{

int size;

cout << "Enter size: ";

cin >> size;

int arr[size];

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

{

arr[i] = rand() % 69;

}

cout << "\nUnsorted array is:";

printarr(arr, size);

insertionSort(arr, size);

cout << "\n";

cout << "\nSorted array is:";

printarr(arr, size);

return 0;

}

1. Heap Sort

#include <iostream>

using namespace std;

void maxheap(int arr[], int size, int root)

{

int max = root;

int left = (2 \* root) + 1;

int right = (2 \* root) + 2;

if (size > left && arr[max] < arr[left])

{

max = left;

}

if (size > right && arr[max] < arr[right])

{

max = right;

}

if (max != root)

{

swap(arr[root], arr[max]);

maxheap(arr, size, max);

}

}

void heapsort(int arr[], int size)

{

for (int i = size / 2 - 1; i >= 0; i--)

{

maxheap(arr, size, i);

}

for (int i = size - 1; i > 0; i--)

{

swap(arr[0], arr[i]);

maxheap(arr, i, 0);

}

}

int main()

{

int n;

cout<<"Enter total size of the array: ";

cin>>n;

int arr1[n];

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

{

cout<<"Enter element "<<i+1<<": ";

cin>>arr1[i];

}

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

{

cout << arr1[i] << " ";

}

cout << "\n";

heapsort(arr1, n);

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

{

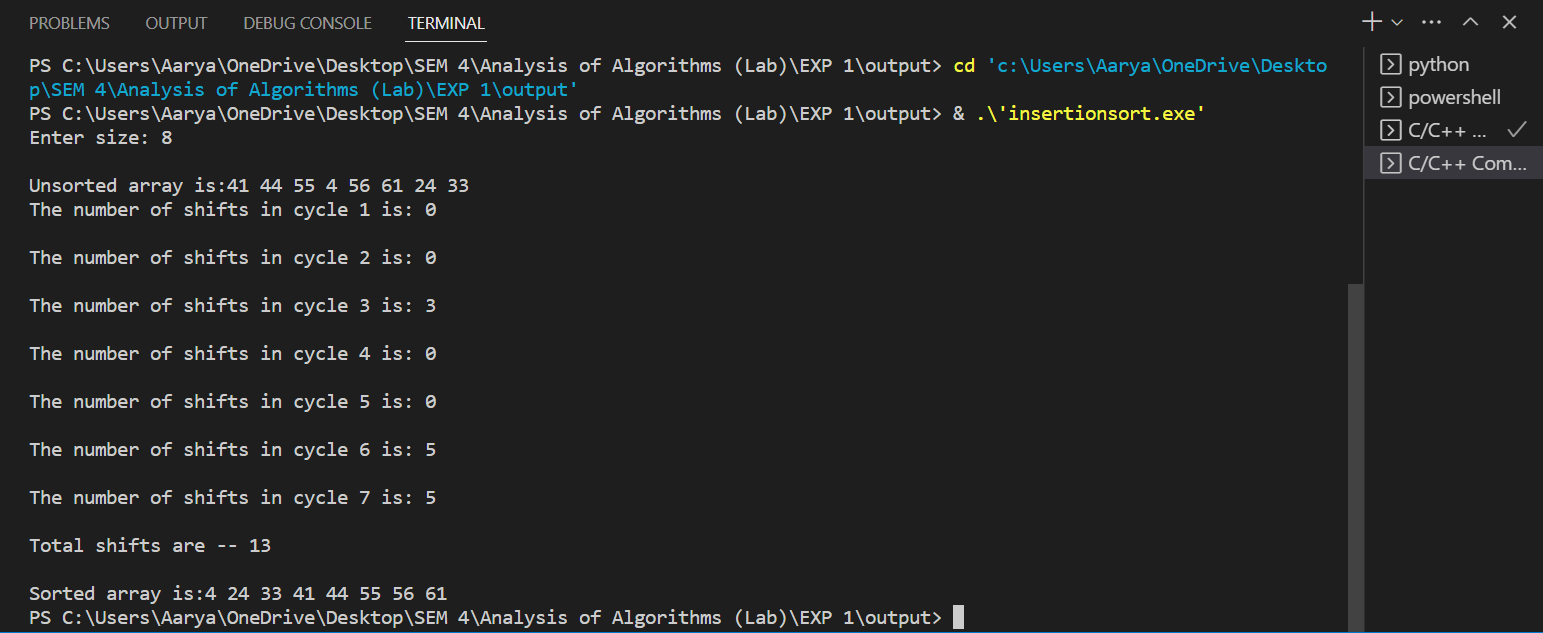
cout << arr1[i] << " ";

}

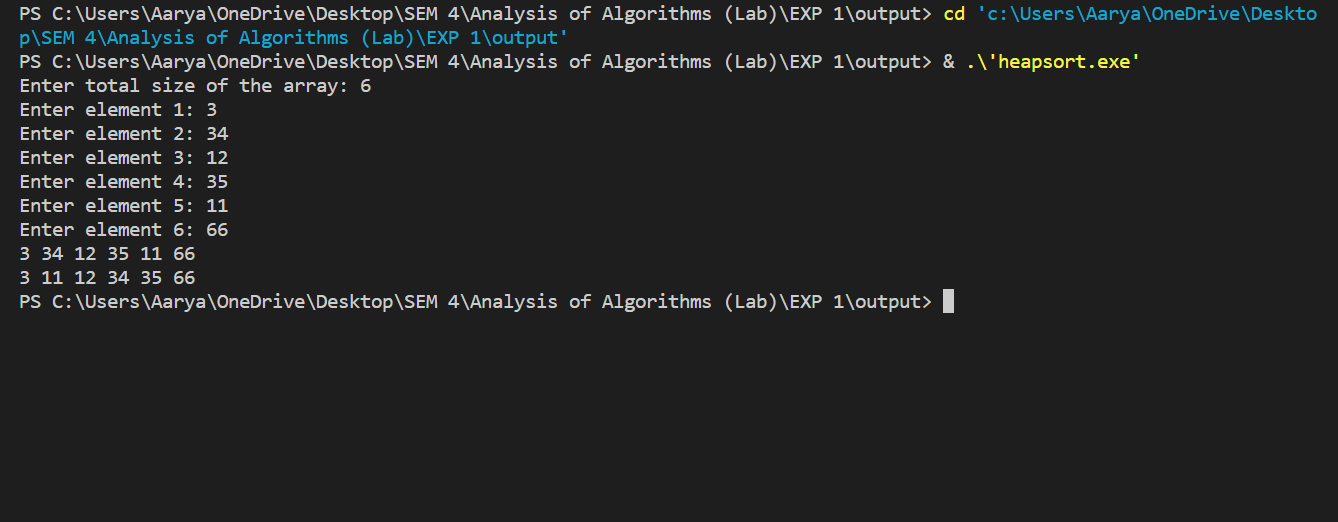
}

**Output(o) of insertion sort & Heap sort:**

1. Insertion Sort



1. Heap Sort



.

**Results:**

**Time Complexity of Insertion sort:**

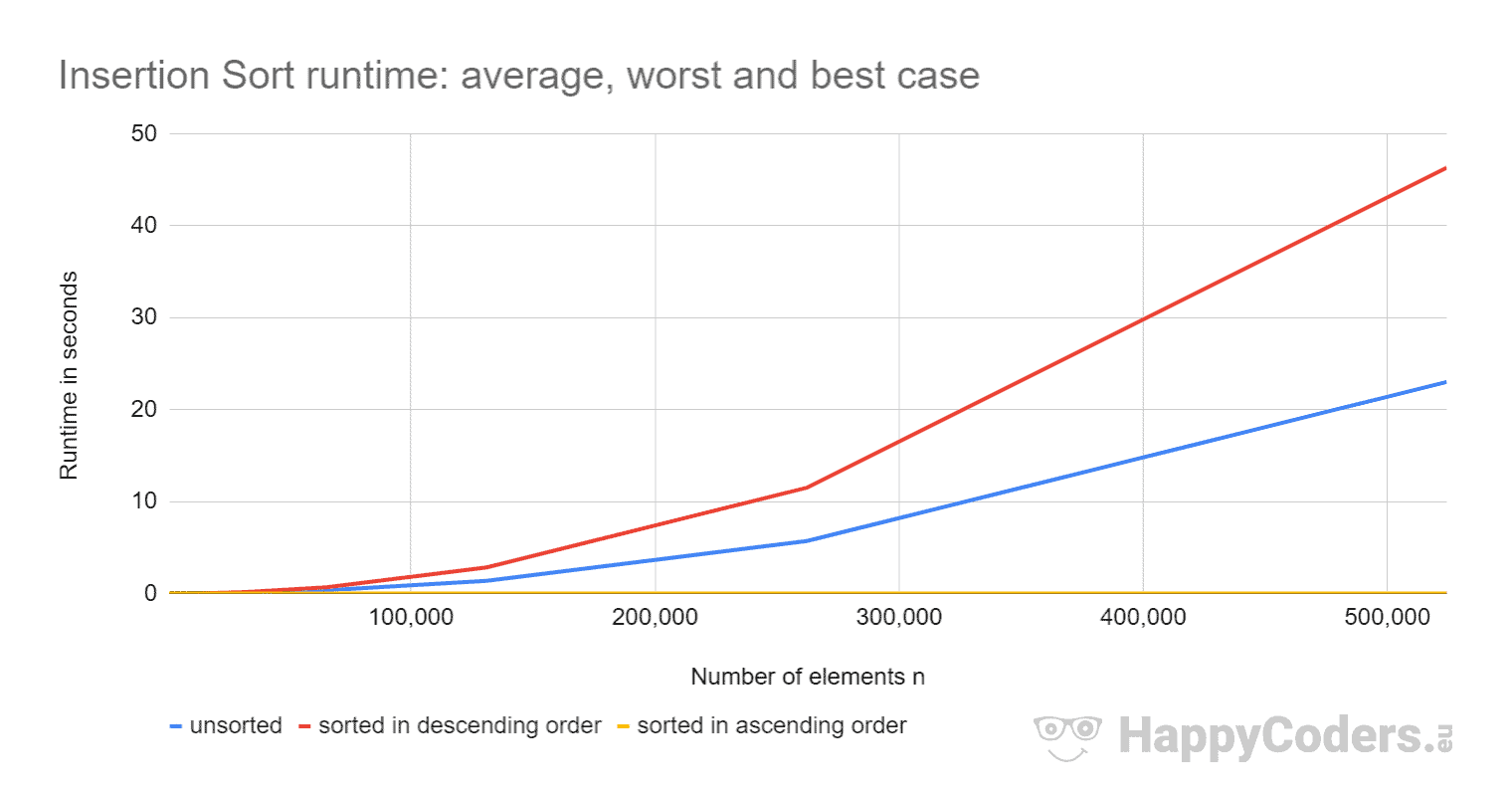
**Worst Case Analysis:**

|  |  |  |  |
| --- | --- | --- | --- |
| **Sr. No.** | **Input size** | **No: of steps from Algorithm analysis** | **No: of steps from Theoretical analysis** |
|  | 5 | 10 | 10 |
|  | 10 | 45 | 45 |
|  | 50 | 1212 | 1212 |
|  | 100 | 4884 | 4884 |

**Best Case Analysis:**

|  |  |  |  |
| --- | --- | --- | --- |
| **Sr. No.** | **Input size** | **No: of steps from Algorithm analysis** | **No: of steps from Theoretical analysis** |
|  | 5 |  |  |
|  | 10 |  |  |
|  | 50 |  |  |
|  | 100 |  |  |

**Graph:-**



**Time Complexity of Heap sort:**

**Worst Case Analysis:**

|  |  |  |  |
| --- | --- | --- | --- |
| **Sr. No.** | **Input size** | **No: of steps from Algorithm analysis** | **No: of steps from Theoretical analysis** |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

**Best Case Analysis:**

|  |  |  |  |
| --- | --- | --- | --- |
| **Sr. No.** | **Input size** | **No: of steps from Algorithm analysis** | **No: of steps from Theoretical analysis** |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

**GRAPH**

**Conclusion: (Based on the observations):**

**Outcome:**

**References:**

1. Richard E. Neapolitan, " Foundation of Algorithms ", 5th Edition 2016, Jones & Bartlett Students Edition
2. Harsh Bhasin , " Algorithms : Design & Analysis", 1st Edition 2013, Oxford Higher education, India
3. T.H. Coreman ,C.E. Leiserson,R.L. Rivest, and C. Stein, " Introduction to algorithms", 3rd Edition 2009, Prentice Hall India Publication
4. Jon Kleinberg, Eva Tardos, " Algorithm Design", 10th Edition 2013, Pearson India Education Services Pvt. Ltd.