BUBT | BANGLADESH UNIVERSITY OF BUSINESS AND TECHNOLOGY

Committed to Academic Excellence

Lab Report-03&04

Course Title	: Algorithms
Course Code	: CSE 242

Submitted By :

Name	: Md. Mursalin Hasan Nirob
ID No	: 21225103423
Intake	: 49
Section	: 10
Program	: B.Sc. Engg. in CSE

Submitted To :

Name : Faria Binte Kader

Lecturer

Department of CSE

Bangladesh University of Business & Technology

Date of Submision:09/08/2023

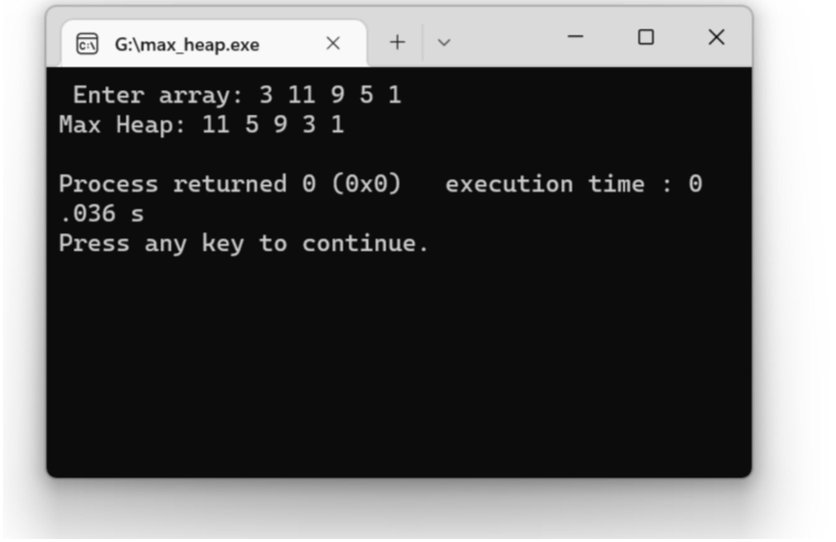
Q- 1: For a given array of unsorted integer numbers of size N, build a Max Heap and print the numbers.

Code:

```
 #include <iostream>

using namespace std;

void heapify(int arr[], int n, int i) {

   int lar = i;

   int left = 2 * i + 1;

   int right = 2 * i + 2;

   if (left < n && arr[left] > arr[lar]) {

      lar = left;

   }

   if (right < n && arr[right] > arr[lar]) {

      lar = right;

   }

   if (lar != i) {

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

      heapify(arr, n, lar);

   }

}

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

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

      heapify(arr, n, i);
```

```
    }
}
void printArray(int arr[], int n) {
    for (int i = 0; i < n; i++) {
        cout << arr[i] << " ";
    }
    cout << endl;
}
int main() {
    int arr[] = {3, 11, 9, 5, 1};
    int n = sizeof(arr) / sizeof(arr[0]);

    cout << " Enter array: ";
    printArray(arr, n);
    buildMaxHeap(arr, n);
    cout << "Max Heap: ";
    printArray(arr, n);
    return 0;
}
```

Output:



```
 Enter array: 3 11 9 5 1
Max Heap: 11 5 9 3 1

Process returned 0 (0x0)   execution time : 0
.036 s
Press any key to continue.
```

Explanation:

A max-heap is a complete binary tree in which the value in each internal node is greater than or equal to the values in the children of that node. The max heap property ensures that the maximum element is always at the top of the heap, allowing for efficient access to this maximum value.

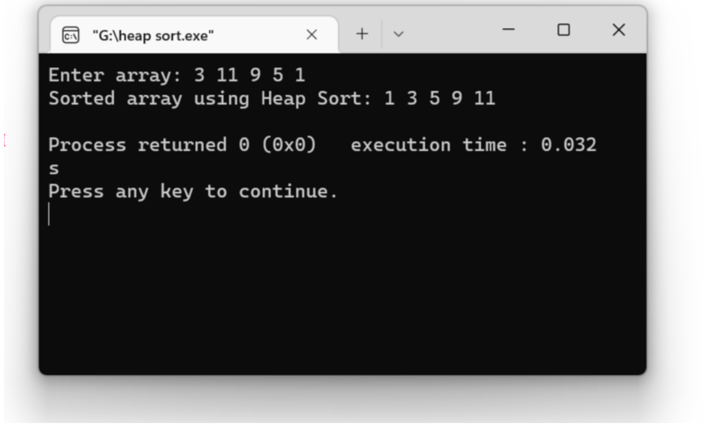
Q- 2: For a given array of unsorted integer numbers of size N, sort them using the Heap Sort algorithm.

Solve:

```
#include <iostream>
using namespace std;

void heapify(int arr[], int n, int i) {
   int lar = i;
   int left = 2 * i + 1;
   int right = 2 * i + 2;

   if (left < n && arr[left] > arr[lar]) {
      lar = left;
   }

   if (right < n && arr[right] > arr[lar]) {
      lar = right;
   }
```

```
    if (lar != i) {
        swap(arr[i], arr[lar]);
        heapify(arr, n, lar);
    }
}

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

    for (int i = n / 2 - 1; i >= 0; i--) {
        heapify(arr, n, i);
    }

    for (int i = n - 1; i > 0; i--) {
        swap(arr[0], arr[i]);
        heapify(arr, i, 0);
    }
}

void printArray(int arr[], int n) {
    for (int i = 0; i < n; i++) {
        cout << arr[i] << " ";
    }
    cout << endl;
}

int main() {
    int arr[] = {3, 11, 9, 5, 1};
    int n = sizeof(arr) / sizeof(arr[0]);

    cout << "Enter array: ";
    printArray(arr, n);

    heapSort(arr, n);

    cout << "Sorted array using Heap Sort: ";
    printArray(arr, n);

    return 0;
}
```

Output:



```
Enter array: 3 11 9 5 1
Sorted array using Heap Sort: 1 3 5 9 11

Process returned 0 (0x0)   execution time : 0.032
s
Press any key to continue.
```

Explanation: Heap Sort is a comparison-based sorting algorithm that uses the properties of a binary heap data structure to sort an array in ascending or descending order. It combines the benefits of the heap data structure's efficient insertion and deletion operations with the advantages of the divide-and-conquer strategy.

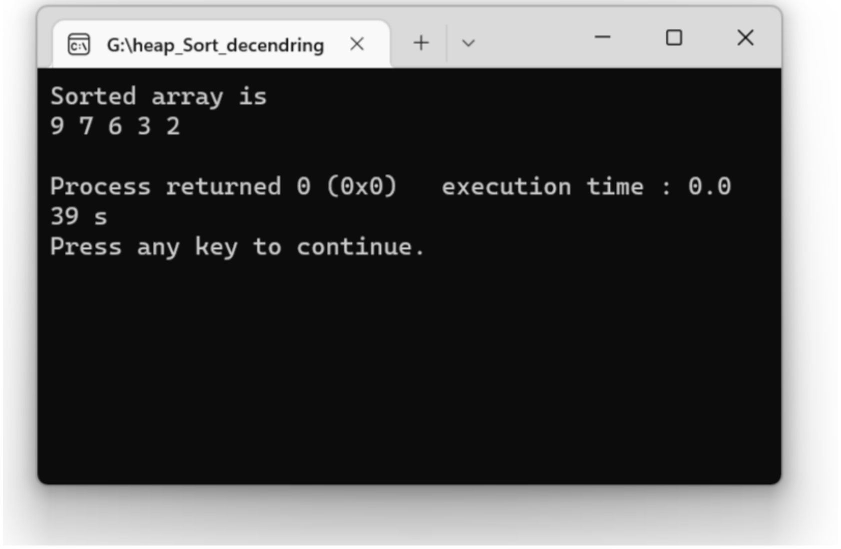
Q-03: For a given array of unsorted integer numbers of size N, sort them using the Heap Sort algorithm in a descending order.

Solve:

```
#include <bits/stdc++.h>
using namespace std;
void heapify(int arr[], int n, int i)
{
	int smallest = i;
	int l = 2 * i + 1;
	int r = 2 * i + 2;


	if (l < n && arr[l] < arr[smallest])
		smallest = l;


	if (r < n && arr[r] < arr[smallest])
		smallest = r;

	if (smallest != i) {
		swap(arr[i], arr[smallest]);
```

```
        heapify(arr, n, smallest);
    }
}


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

    for (int i = n / 2 - 1; i >= 0; i--)
        heapify(arr, n, i);


    for (int i = n - 1; i >= 0; i--) {

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


        heapify(arr, i, 0);
    }
}


void printArray(int arr[], int n)
{
    for (int i = 0; i < n; ++i)
        cout << arr[i] << " ";
    cout << "\n";
}


int main()
{
    int arr[] = { 7, 6, 3, 2, 9 };
    int n = sizeof(arr) / sizeof(arr[0]);

    heapSort(arr, n);

    cout << "Sorted array is \n";
    printArray(arr, n);
}
```

Output:



```
Sorted array is
9 7 6 3 2

Process returned 0 (0x0)   execution time : 0.0
39 s
Press any key to continue.
```

Explanation: Heap Sort in descending order follows the same basic steps as Heap Sort in ascending order, with a slight modification. Instead of building a max heap, where the largest element is at the root and the children are smaller, we build a min heap. In the min heap, the smallest element is at the root, and the children are larger.

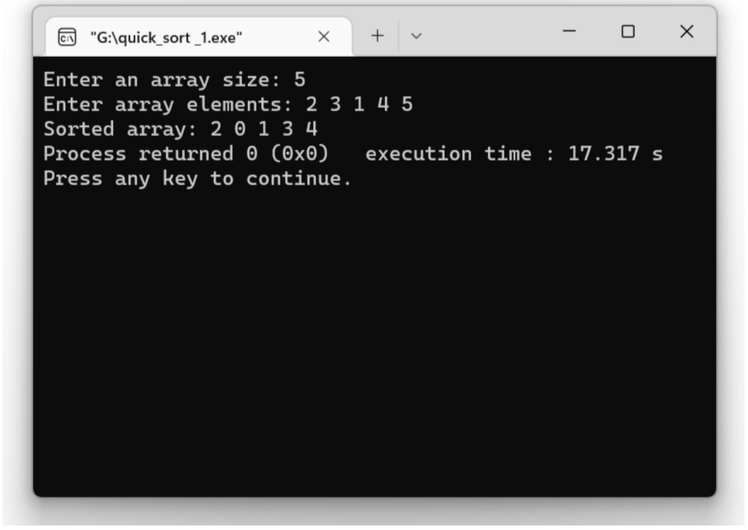
Q-04: For a given array of unsorted integer numbers of size N, sort them using the Quick Sort algorithm.

Code:

```
#include<iostream>
using namespace std;
int partition(int arr[],int l, int r)
{
   int pivot = arr[r];
   int i = l-1;
   for(int j = l;j<r;j++)
   {
      if(arr[j]<pivot)
      {
         i++;
         swap(arr[i],arr[j]);
      }
   }
   swap(arr[i+1],arr[r]);
   return i+1;
}
```

```
void quickSort(int arr[],int l, int r)
{
   if(l<r)
   {
      int p = partition(arr,l,r);
      quickSort(arr,l,p-1);
      quickSort(arr,p+1,r);
   }
}
int main()
{
   int n;
   cout<<"Enter an array size: ";
   cin>>n;
   int arr[n];
   cout<<"Enter array elements: ";
   for(int i = 0; i<n;i++)
   {
      cin>>arr[i];
   }
   quickSort(arr,1,5);
   cout<<"Sorted array: ";
   for(int i = 0; i<5;i++)
   {
      cout<<arr[i]<<" ";
   }
}
```

Output:



```
Enter an array size: 5
Enter array elements: 2 3 1 4 5
Sorted array: 2 0 1 3 4
Process returned 0 (0x0)   execution time : 17.317 s
Press any key to continue.
```

<u>Explanation:</u> Quick Sort is known for its efficiency and is often faster in practice compared to other sorting algorithms like Bubble Sort and Insertion Sort. The algorithm works by selecting a pivot element from the array and partitioning the other elements into two sub-arrays, according to whether they are less than or greater than the pivot. The sub-arrays are then sorted recursively. This process of partitioning and sorting continues until the entire array is sorted.

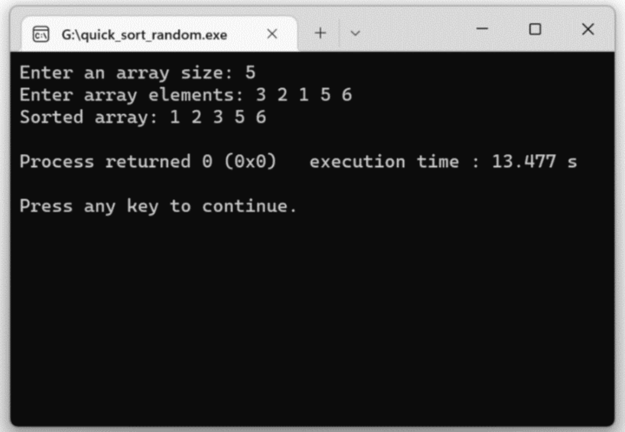
<u>Q-05</u>: For a given array of unsorted integer numbers of size N, sort them using the Randomized Quick Sort algorithm.

<u>Solve:</u>

```
 #include <iostream>
#include <cstdlib>
#include <ctime>

using namespace std;

int partition(int arr[], int l, int r)
{

   int randomIndex = l + rand() % (r - l + 1);
   swap(arr[randomIndex], arr[r]);

   int pivot = arr[r];
   int i = l - 1;

   for (int j = l; j < r; j++)
   {
      if (arr[j] < pivot)
      {
         i++;
         swap(arr[i], arr[j]);
      }
   }

   swap(arr[i + 1], arr[r]);
   return i + 1;
}

void quickSort(int arr[], int l, int r)
{
   if (l < r)
   {
      int p = partition(arr, l, r);
      quickSort(arr, l, p - 1);
```

```
        quickSort(arr, p + 1, r);
    }
}

int main()
{
    srand(time(0));
    int n;
    cout << "Enter an array size: ";
    cin >> n;
    int arr[n];
    cout << "Enter array elements: ";
    for (int i = 0; i < n; i++)
    {
        cin >> arr[i];
    }
    quickSort(arr, 0, n - 1);
    cout << "Sorted array: ";
    for (int i = 0; i < n; i++)
    {
        cout << arr[i] << " ";
    }
    cout << endl;
}
```

Output:



```
Enter an array size: 5
Enter array elements: 3 2 1 5 6
Sorted array: 1 2 3 5 6

Process returned 0 (0x0)   execution time : 13.477 s

Press any key to continue.
```

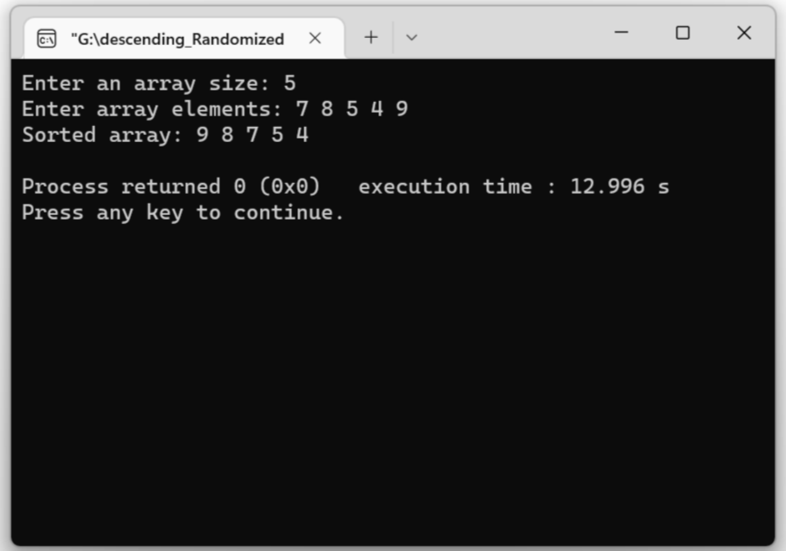
Explanation: Randomized Quick Sort seeks to mitigate this problem by introducing an element of randomness into the pivot selection process. Instead of always choosing a fixed pivot element, Randomized Quick Sort selects the pivot randomly from the array. This randomness helps distribute the chance of selecting a bad pivot, which in turn improves the average-case and expected performance of the algorithm.

Q-06: For a given array of unsorted integer numbers of size N, sort them in a descending order using the Randomized Quick Sort algorithm.

Solve:

```
#include <iostream>
#include <cstdlib>
#include <ctime>

using namespace std;

int partition(int arr[], int l, int r)
{

   int randomIndex = l + rand() % (r - l + 1);
   swap(arr[randomIndex], arr[r]);

   int pivot = arr[r];
   int i = l - 1;

   for (int j = l; j < r; j++)
   {
      if (arr[j] > pivot)
      {
         i++;
         swap(arr[i], arr[j]);
      }
   }

   swap(arr[i + 1], arr[r]);
   return i + 1;
}

void quickSort(int arr[], int l, int r)
{
   if (l < r)
   {
      int p = partition(arr, l, r);
      quickSort(arr, l, p - 1);
      quickSort(arr, p + 1, r);
   }
}

int main()
{
   srand(time(0));
   int n;
   cout << "Enter an array size: ";
   cin >> n;
   int arr[n];
   cout << "Enter array elements: ";
   for (int i = 0; i < n; i++)
   {
      cin >> arr[i];
   }
   quickSort(arr, 0, n - 1);
   cout << "Sorted array: ";
   for (int i = 0; i < n; i++)
   {
      cout << arr[i] << " ";
   }
   cout << endl;
}
```

Output:



```
Enter an array size: 5
Enter array elements: 7 8 5 4 9
Sorted array: 9 8 7 5 4

Process returned 0 (0x0)   execution time : 12.996 s
Press any key to continue.
```

Explanation:

Descending order using Randomized Quick Sort involves sorting an array of elements in such a way that the largest elements come first, with the use of the

Randomized Quick Sort algorithm. This is achieved by modifying the partitioning process of Quick Sort so that elements greater than the pivot are placed to the left of the pivot, and elements smaller than the pivot are placed to the right. The pivot selection remains randomized to improve average-case performance.