قسمت 1)

در این قسمت ابتدا تمام الگوریتم های سورت را می نویسیم سپس برای هر یک، یک main نوشته و با اجرای ان فایل الگوریتم را تست می کنیم:

1) Merge Sort:

```
void merge_sort(int *arr, int low, int high, int n)
{
    int mid;
    if (low < high){
        //divide the array at mid and sort independently using merge sort
        mid=(low+high)/2;
        merge_sort(arr,low,mid,n);
        merge_sort(arr,mid+1,high,n);
        //merge or conquer sorted arrays
        merge(arr,low,high,mid,n);
    }
}
```

```
void merge(int *arr, int low, int high, int mid,int n)
{
    int i, j, k, c[n];
    i = low;
    k = low;
    j = mid + 1;
    while (i <= mid && j <= high) {
        if (arr[i] < arr[j]) {
            c[k] = arr[i];
            k++;
            i++;
        }
        else  {
            c[k] = arr[j];
            k++;
            j++;
        }
    }
    while (i <= mid) {
        c[k] = arr[i];
        k++;
        i++;
    }
    while (j <= high) {
        c[k] = arr[j];
        k++;
        j++;
    }
    for (i = low; i < k; i++)  {
        arr[i] = c[i];
    }
}
```

```
int main()
{
    int myarray[30], num;
    cout<<"Enter number of elements to be sorted:";
    cin>>num;
    cout<<"Enter "<<num<<" elements to be sorted:";
    for (int i = 0; i < num; i++) { cin>>myarray[i];
    }
    merge_sort(myarray, 0, num-1);
    cout<<"Sorted array\n";
    for (int i = 0; i < num; i++)
    {
        cout<<myarray[i]<<"\t";
    }
}
```

2) Heap Sort:

```
void heapify(int array[], int len, int i)
{
    int largest = i;
    int left = 2 * i + 1;
    int right = 2 * i + 2;

    if (left < len && array[left] > array[largest])
        largest = left;

    if (right < len && array[right] > array[largest])
        largest = right;

    if (largest != i)
    {
        swap(array[i], array[largest]);

        heapify(array, len, largest);
    }
}

void heapSort(int array[], int len)
{
    for (int i = len / 2 - 1; i >= 0; i--)
        heapify(array, len, i);

    for (int i = len - 1; i >= 0; i--)
    {
        swap(array[0], array[i]);
        heapify(array, i, 0);
    }
}
```

```
void print(int array[], int len)
{
    cout << "After sorting the array is: \n";
    for (int i = 0; i < len; ++i)
        cout << array[i] << " ";
    cout << endl;
}

int main()
{
    int array[] = {12, 20, 6, 15, 2, 11,123};
    int len = sizeof(array) / sizeof(array[0]);

    heapSort(array, len);
    print(array, len);
}
```

3) Quick Sort:

```
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];    // pivot
    int i = (low - 1);  // Index of smaller element

    for (int j = low; j <= high- 1; j++)
    {
        // If current element is smaller than or
        // equal to pivot
        if (arr[j] <= pivot)
        {
            i++;    // increment index of smaller element
            swap(&arr[i], &arr[j]);
        }
    }
    swap(&arr[i + 1], &arr[high]);
    return (i + 1);
}
```

```
void quickSort(int arr[], int low, int high)
{
    if (low < high)
    {
        /* pi is partitioning index, arr[p] is now
           at right place */
        int pi = partition(arr, low, high);

        // Separately sort elements before
        // partition and after partition
        quickSort(arr, low, pi - 1);
        quickSort(arr, pi + 1, high);
    }
}
```

```c
void printArray(int arr[], int size)
{
    int i;
    for (i=0; i < size; i++)
        printf("%d ", arr[i]);
    printf("n");
}

// Driver program to test above functions
int main()
{
    int arr[] = {10, 7, 8, 9, 1, 5};
    int n = sizeof(arr)/sizeof(arr[0]);
    quickSort(arr, 0, n-1);
    printf("Sorted array: n");
    printArray(arr, n);
    return 0;
}
```

4) Shell Sort:

```
int shellSort(int arr[], int n)
{
    // Start with a big gap, then reduce the gap
    for (int gap = n/2; gap > 0; gap /= 2)
    {
        // Do a gapped insertion sort for this gap size.
        // The first gap elements a[0..gap-1] are already in gapped order
        // keep adding one more element until the entire array is
        // gap sorted
        for (int i = gap; i < n; i += 1)
        {
            // add a[i] to the elements that have been gap sorted
            // save a[i] in temp and make a hole at position i
            int temp = arr[i];

            // shift earlier gap-sorted elements up until the correct
            // location for a[i] is found
            int j;
            for (j = i; j >= gap && arr[j - gap] > temp; j -= gap)
                arr[j] = arr[j - gap];

            //  put temp (the original a[i]) in its correct location
            arr[j] = temp;
        }
    }
    return 0;
}
```

```
int main()
{
    int arr[] = {12, 34, 54, 2, 3}, i;
    int n = sizeof(arr)/sizeof(arr[0]);

    cout << "Array before sorting: \n";
    printArray(arr, n);

    shellSort(arr, n);

    cout << "\nArray after sorting: \n";
    printArray(arr, n);

    return 0;
}
```

5) Insertion Sort:

```
void insertionSort(int arr[], int n)
{
    int i, key, j;
    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[] = { 12, 11, 13, 5, 6 };
    int n = sizeof(arr) / sizeof(arr[0]);

    insertionSort(arr, n);
    printArray(arr, n);

    return 0;
}
```

قسمت 2)

سپس در این قسمت تمام سورت ها را در یک فایل قرار می دهیم تا آنها را به ترتیب تست کنیم.
بعد از ان یک تابع می نویسیم تا با گرفتن یک عدد ورودی یک ارایه Uniform با ان تعداد عضو بسازد پس داریم:

```
/************************ Creating Array Function ************************/
void CreateArray(int* RandomArray, int n)
{
 srand((unsigned)time(NULL));
 int newArray[n];

 for (int i = 0; i < n; i++){
      int b = rand() ;
      newArray[i] = b;
  }
  RandomArray = newArray;
}
/************************ Creating Array Function ************************/
```

سپس یک main نوشته و در ان 5 ارایه با استفاده از تابع ساخته شده می سازیم و ان ها را به عنوان ورودی تابع های سورت می گذاریم.

برای بدست اوردن زمان اجرای هر دستور از کتابخانه ی

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

Chrono استفاده کرده ایم.

```
    int MyArray1[n];
    int MyArray2[n];
    int MyArray3[n];
    int MyArray4[n];
    int MyArray5[n];
    CreateArray(MyArray1, n);
    CreateArray(MyArray2, n);
    CreateArray(MyArray3, n);
    CreateArray(MyArray4, n);
    CreateArray(MyArray5, n);
```

سپس برای هر کدام از سورت ها این ورودی ها را می دهیم پس داریم:

```
/************************* MergeSort *********************************/
cout<<"Merge Sort Timing\n";
// Get starting timepoint
auto startMerge = high_resolution_clock::now();
//MergeSort Function
merge_sort(MyArray1 ,0, n-1 , n);
// Get ending timepoint
auto stopMerge = high_resolution_clock::now();

// use duration cast method
auto durationMerge = duration_cast<microseconds>(stopMerge - startMerge);
cout << "Time taken by MergeSort Function: " << durationMerge.count() << " microseconds" << endl;
```

```
/************************* HeapSort *********************************/
cout << "Heap Sort Timing" << '\n';
// Get starting timepoint
auto startHeap = high_resolution_clock::now();
//MergeSort Function
heapSort(MyArray2, n);
// Get ending timepoint
auto stopHeap = high_resolution_clock::now();

// use duration cast method
auto durationHeap = duration_cast<microseconds>(stopHeap - startHeap);
cout << "Time taken by HeapSort Function: " << durationHeap.count() << " microseconds" << endl;
```

```
/************************* QuickSort *********************************/
cout << "Quick Sort Timing" << '\n';
// Get starting timepoint
auto startQuick = high_resolution_clock::now();
//MergeSort Function
quicksort(MyArray3, 0, n-1);
// Get ending timepoint
auto stopQuick = high_resolution_clock::now();

// use duration cast method
auto durationQuick = duration_cast<microseconds>(stopQuick - startQuick);
cout << "Time taken by QuickSort Function: " << durationQuick.count() << " microseconds" << endl;
```

```
/************************* ShellSort *********************************/
cout << "Shell Sort Timing" << '\n';
// Get starting timepoint
auto startShell = high_resolution_clock::now();
//MergeSort Function
shellSort(MyArray4, n);
// Get ending timepoint
auto stopShell = high_resolution_clock::now();

// use duration cast method
auto durationShell = duration_cast<microseconds>(stopShell - startShell);
cout << "Time taken by ShellSort Function: " << durationShell.count() << " microseconds" << endl;
```

```
/************************ InsertionSort ********************************/
cout << "Insertion Sort Timing" << '\n';
// Get starting timepoint
auto startInsertion = high_resolution_clock::now();
//MergeSort Function
shellSort(MyArray5, n);
// Get ending timepoint
auto stopInsertion = high_resolution_clock::now();

// use duration cast method
auto durationInsertion = duration_cast<microseconds>(stopInsertion - startInsertion);
cout << "Time taken by InsertionSort Function: " << durationInsertion.count() << " microseconds" << endl;
```

در این حالت زمان اجرای هر یک را بدست می آوریم.

برای حالت ورودی سورت شده ابتدا ارایه ی ساخته شده را سورت می کنیم و سپس ان را به عنوان ورودی به تابع های سورت می دهیم:

```
//UnComment for Sorted Input
heapSort(MyArray1, n);
heapSort(MyArray2, n);
heapSort(MyArray3, n);
heapSort(MyArray4, n);
heapSort(MyArray5, n);
```

برای حالت ورودی Reverse Sort مانند زیر ابتدا ورودی ها را سورت کرده سپس ارایه را بر عکس می کنیم و سپس ان ها را به عنوان ورودی به تابع های سورت می دهیم:

```
//UnComment for Reverse Sort Input
heapSort(MyArray1, n);
heapSort(MyArray2, n);
heapSort(MyArray3, n);
heapSort(MyArray4, n);
heapSort(MyArray5, n);
for(int i=0;i<=5;i++){
  MyArray1[i] = MyArray1[n-i];
  MyArray2[i] = MyArray2[n-i];
  MyArray3[i] = MyArray3[n-i];
  MyArray4[i] = MyArray4[n-i];
  MyArray5[i] = MyArray5[n-i];
```

برای گرفتن تعداد عضو های ارایه از ورودی نیز مانند زیر عمل می کنیم:

```
//Get Input From User
cout << "Enter number of elements to be sorted:" << endl;
cin >> n;
```

و از n به عنوان تعداد عضو های ارایه استفاده می کنیم

بعد از اجرای کد زمان اجرا باری حالات مختلف به شکل زیر است:

```
Input Number Is:50000
Merge Sort Timing
Time taken by MergeSort Function: 5933 microseconds
Heap Sort Timing
Time taken by HeapSort Function: 1263 microseconds
Quick Sort Timing
Time taken by QuickSort Function: 4586182 microseconds
Shell Sort Timing
Time taken by ShellSort Function: 2788 microseconds
Insertion Sort Timing
Time taken by InsertionSort Function: 2812 microseconds
```

Figure 1:50K Normal

```
Input Number Is:100000
Merge Sort Timing
Time taken by MergeSort Function: 10290 microseconds
Heap Sort Timing
Time taken by HeapSort Function: 2357 microseconds
Quick Sort Timing
Time taken by QuickSort Function: 6280602 microseconds
Shell Sort Timing
Time taken by ShellSort Function: 12901 microseconds
Insertion Sort Timing
Time taken by InsertionSort Function: 14897 microseconds
```

Figure 2:100K Normal

```
Input Number Is:150000
Merge Sort Timing
Time taken by MergeSort Function: 24872 microseconds
Heap Sort Timing
Time taken by HeapSort Function: 18429 microseconds
Quick Sort Timing
Time taken by QuickSort Function: 14122583 microseconds
Shell Sort Timing
Time taken by ShellSort Function: 16258 microseconds
Insertion Sort Timing
Time taken by InsertionSort Function: 15500 microseconds
```

Figure 3:150K Normal

```
Input Number Is:200000
Merge Sort Timing
Time taken by MergeSort Function: 21401 microseconds
Heap Sort Timing
Time taken by HeapSort Function: 35252 microseconds
Quick Sort Timing
Time taken by QuickSort Function: 25376372 microseconds
Shell Sort Timing
Time taken by ShellSort Function: 21331 microseconds
Insertion Sort Timing
Time taken by InsertionSort Function: 26952 microseconds
```

Figure 4:200K Normal

:□□□□□ □□□ □□□□ □□□□□ □□□□ □□□□

```
Input Number Is:50000
Merge Sort Timing
Time taken by MergeSort Function: 5247 microseconds
Heap Sort Timing
Time taken by HeapSort Function: 971 microseconds
Quick Sort Timing
Time taken by QuickSort Function: 4499352 microseconds
Shell Sort Timing
Time taken by ShellSort Function: 2622 microseconds
Insertion Sort Timing
Time taken by InsertionSort Function: 2604 microseconds
```

Figure 5:50K Sorted

```
Input Number Is:100000
Merge Sort Timing
Time taken by MergeSort Function: 9933 microseconds
Heap Sort Timing
Time taken by HeapSort Function: 2059 microseconds
Quick Sort Timing
Time taken by QuickSort Function: 17975150 microseconds
Shell Sort Timing
Time taken by ShellSort Function: 5535 microseconds
Insertion Sort Timing
Time taken by InsertionSort Function: 5725 microseconds
```

Figure 6:100K Sorted

```
Input Number Is:150000
Merge Sort Timing
Time taken by MergeSort Function: 16245 microseconds
Heap Sort Timing
Time taken by HeapSort Function: 3633 microseconds
Shell Sort Timing
Time taken by ShellSort Function: 9088 microseconds
Insertion Sort Timing
Time taken by InsertionSort Function: 9445 microseconds
```

Figure 7:150K Sorted

```
Input Number Is:200000
Merge Sort Timing
Time taken by MergeSort Function: 21792 microseconds
Heap Sort Timing
Time taken by HeapSort Function: 4116 microseconds
Shell Sort Timing
Time taken by ShellSort Function: 12484 microseconds
Insertion Sort Timing
Time taken by InsertionSort Function: 11883 microseconds
```

Figure 8:200K Sorted

:□□□□□ □□□ □□ □□□ □□□□ □□□□□ □□□□ □□□□ □□□□

```
Input Number Is:50000
Merge Sort Timing
Time taken by MergeSort Function: 5349 microseconds
Heap Sort Timing
Time taken by HeapSort Function: 1020 microseconds
Quick Sort Timing
Time taken by QuickSort Function: 4474829 microseconds
Shell Sort Timing
Time taken by ShellSort Function: 2616 microseconds
Insertion Sort Timing
Time taken by InsertionSort Function: 2634 microseconds
```

Figure 9:50K Reverse

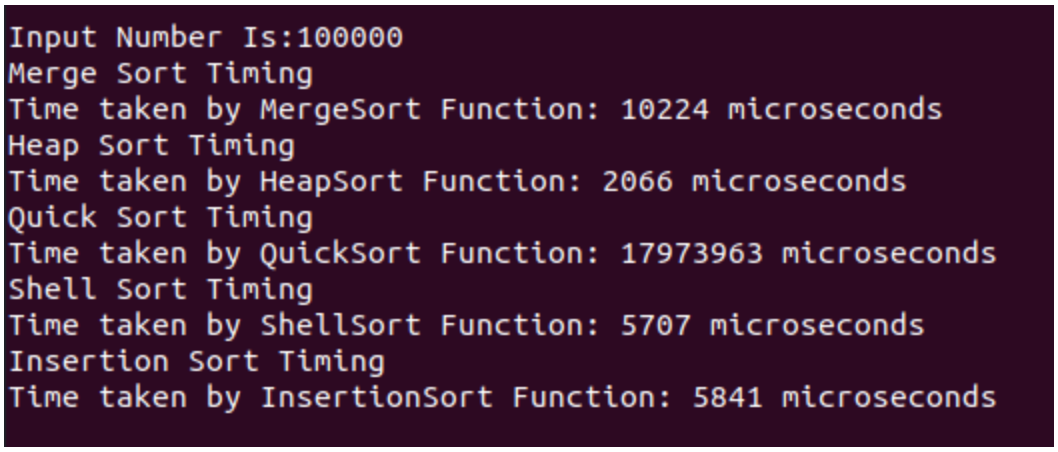

Figure 10:100K Reverse

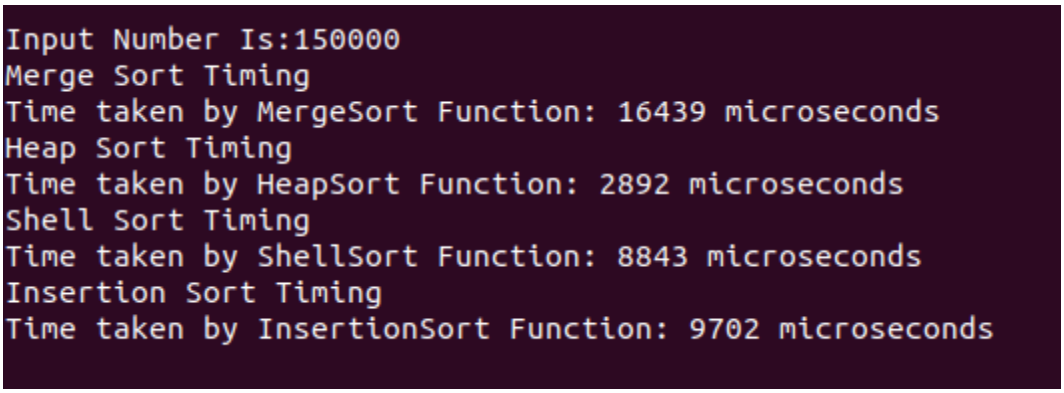

Figure 11:150K Reverse

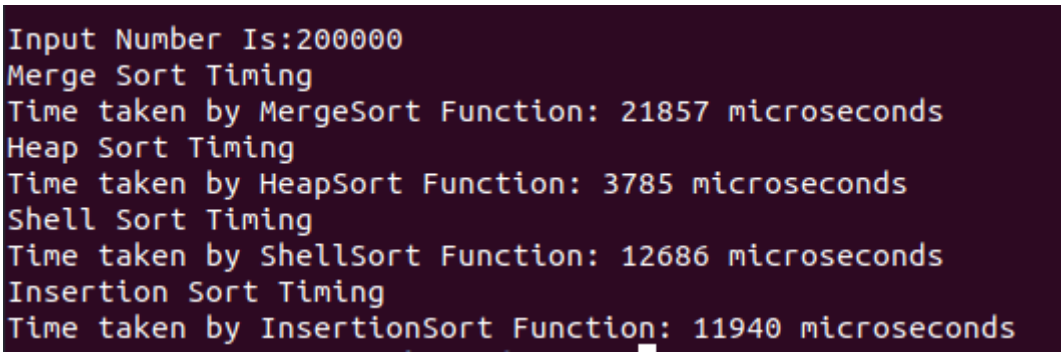

Figure 12:200K Reverse

سهیل شیروانی
810195416

به نام خدا

DS_Ca1

بعد از اجرای کد ها برای زمان اجرای آنها جدول های زیر را داریم:

Sorted

	50K Input	100K Input	150K Input	200K Input
Merge Sort	5247	9933	16245	21792
Heap Sort	971	2059	3633	4116
Quick Sort	4499352	17975150		
Shell Sort	2622	5535	9088	12484
Insertion Sort	2604	5725	9445	11883

Reverse Sort

	50K Input	100K Input	150K Input	200K Input
Merge Sort	5312	10224	16439	21857
Heap Sort	976	2066	2892	3785
Quick Sort	4474829	17973963		
Shell Sort	2719	5707	8843	12686
Insertion Sort	2705	5841	9702	11940

Normal

	50K Input	100K Input	150K Input	200K Input
Merge Sort	5933	10290	24872	21401
Heap Sort	1263	2357	18429	35252
Quick Sort	4586182	6280602	14122583	25376372
Shell Sort	2788	12901	16258	21331
Insertion Sort	2812	14897	15500	26952

برای درجه هر کدام از Sort ها داریم:

$$Merge\ Sort:\ \theta(nLogn)\ in\ all\ 3\ cases\ (worst, average\ and\ best)$$

$$Heap\ Sort: \theta(nLogn)\ in\ all\ 3\ cases\ (worst, average\ and\ best)$$

$$Quick\ Sort: \theta(n2)(Wrost\ case)\ \theta(nLogn)(Best\ Case)\ \theta(\text{nLogn})(\text{Averge Case})$$

$$Shell\ Sort: \text{O}(n2) in\ all\ 3\ cases\ (worst, average\ and\ best)$$

$$Insertion\ Sort: \text{O}(n2)\ in\ all\ 3\ cases\ (worst, average\ and\ best)$$

Curve Fitting

در این قسمت اطلاعات در جدول را به Matlab می دهیم و جدول زیر را بدست می آوریم: