Assignment-4

Ques:

Write a C Program to analyse the time complexity (make a comparative analysis) analysis of the following sorting algorithms

· Insertion Sort

· Bubble Sort

· Selection Sort

· Merge Sort

· Heap Sort

· Shell Sort

and also mention your explanation for the best sorting algorithm.

Code:

#include <stdio.h>

#include <stdlib.h>

#include <time.h>

void swap(int \*xp, int \*yp)

{

    int temp = \*xp;

    \*xp = \*yp;

    \*yp = temp;

}

void bubble\_sort(int v[], int n, FILE \*fptr)

{

    int i, j;

    clock\_t start, end;

    double runtime;

    start = clock();

    for (i = 0; i < n; i++)

    {

        for (j = 0; j < n - i + 1; j++)

        {

            if (v[j] > v[j + 1])

            {

                int temp = v[j];

                v[j] = v[j + 1];

                v[j + 1] = temp;

            }

        }

    }

    end = clock();

    runtime = ((double)(end - start)) / CLOCKS\_PER\_SEC;

    fprintf(fptr, "%d %lf\n", n, runtime);

}

void selection\_sort(int v[], int n, FILE \*fptr)

{

    clock\_t start, end;

    double runtime;

    start = clock();

    int i, j, min\_idx;

    for (i = 0; i < n - 1; i++)

    {

        min\_idx = i;

        for (j = i + 1; j < n; j++)

            if (v[j] < v[min\_idx])

                min\_idx = j;

        if (min\_idx != i)

            swap(&v[min\_idx], &v[i]);

    }

    end = clock();

    runtime = ((double)(end - start)) / CLOCKS\_PER\_SEC;

    fprintf(fptr, "%d %lf\n", n, runtime);

}

void insertion\_sort(int arr[], int n, FILE \*fptr)

{

    clock\_t start, end;

    double runtime;

    start = clock();

    int i, key, j;

    for (i = 1; i < n; i++)

    {

        key = arr[i];

        j = i - 1;

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

        {

            arr[j + 1] = arr[j];

            j = j - 1;

        }

        arr[j + 1] = key;

    }

    end = clock();

    runtime = ((double)(end - start)) / CLOCKS\_PER\_SEC;

    fprintf(fptr, "%d %lf\n", n, runtime);

}

void merge(int arr[], int l, int m, int r)

{

    int i, j, k;

    int n1 = m - l + 1;

    int n2 = r - m;

    int L[n1], R[n2];

    for (i = 0; i < n1; i++)

        L[i] = arr[l + i];

    for (j = 0; j < n2; j++)

        R[j] = arr[m + 1 + j];

    i = 0;

    j = 0;

    k = l;

    while (i < n1 && j < n2)

    {

        if (L[i] <= R[j])

        {

            arr[k] = L[i];

            i++;

        }

        else

        {

            arr[k] = R[j];

            j++;

        }

        k++;

    }

    while (i < n1)

    {

        arr[k] = L[i];

        i++;

        k++;

    }

    while (j < n2)

    {

        arr[k] = R[j];

        j++;

        k++;

    }

}

void mergeSort(int arr[], int l, int r)

{

    if (l < r)

    {

        int m = l + (r - l) / 2;

        mergeSort(arr, l, m);

        mergeSort(arr, m + 1, r);

        merge(arr, l, m, r);

    }

}

void printmerge(int arr[], int n, FILE \*fptr)

{

    clock\_t start, end;

    double runtime;

    start = clock();

    mergeSort(arr, 0, n - 1);

    end = clock();

    runtime = ((double)(end - start)) / (double)CLOCKS\_PER\_SEC;

    fprintf(fptr, "%d %.9lf\n", n, runtime);

}

void maxheapify(int A[], int n, int i)

{

    int l = 2 \* i + 1;

    int r = 2 \* i + 2;

    int largest = i;

    if (l < n && A[l] > A[largest])

        largest = l;

    if (r < n && A[r] > A[l])

        largest = r;

    if (largest != i)

    {

        swap(&A[i], &A[largest]);

        maxheapify(A, n, largest);

    }

}

void buildmaxheap(int a[], int n)

{

    int i;

    for (i = n / 2 - 1; i >= 0; i--)

    {

        maxheapify(a, n, i);

    }

}

void heapsort(int a[], int n)

{

    int i;

    int t = n;

    buildmaxheap(a, n);

    for (i = n - 1; i >= 1; i--)

    {

        swap(&a[0], &a[i]);

        maxheapify(a, i, 0);

    }

}

void printheap(int arr[], int n, FILE \*fptr)

{

    clock\_t start, end;

    double runtime;

    start = clock();

    heapsort(arr, n);

    end = clock();

    runtime = ((double)(end - start)) / (double)CLOCKS\_PER\_SEC;

    fprintf(fptr, "%d %.9lf\n", n, runtime);

}

void shellSort(int arr[], int n)

{

    int gap, i;

    for (gap = n / 2; gap > 0; gap /= 2)

    {

        for (i = gap; i < n; i += 1)

        {

            int temp = arr[i];

            int j;

            for (j = i; j >= gap && arr[j - gap] > temp; j -= gap)

                arr[j] = arr[j - gap];

            arr[j] = temp;

        }

    }

}

void printshell(int arr[], int n, FILE \*fptr)

{

    clock\_t start, end;

    double runtime;

    start = clock();

    shellSort(arr, n);

    end = clock();

    runtime = ((double)(end - start)) / (double)CLOCKS\_PER\_SEC;

    fprintf(fptr, "%d %.9lf\n", n, runtime);

}

int main()

{

    FILE \*fptr1 = fopen("bubble.txt", "w");

    FILE \*fptr2 = fopen("selection.txt", "w");

    FILE \*fptr3 = fopen("insertion.txt", "w");

    FILE \*fptr4 = fopen("merge.txt", "w");

    FILE \*fptr5 = fopen("shell.txt", "w");

    FILE \*fptr6 = fopen("heap.txt", "w");

    int n, i;

    for (n = 1000; n <= 100000; n += 1000)

    {

        int lb = 1, ub = 1000000;

        int arr[n];

        for (i = 0; i < n; i++)

        {

            arr[i] = rand() % (ub - lb + 1) + lb;

        }

        bubble\_sort(arr, n, fptr1);

        selection\_sort(arr, n, fptr2);

        insertion\_sort(arr, n, fptr3);

        printmerge(arr, n, fptr4);

        printshell(arr, n, fptr5);

        printheap(arr, n, fptr6);

    }

    fclose(fptr1);

    fclose(fptr2);

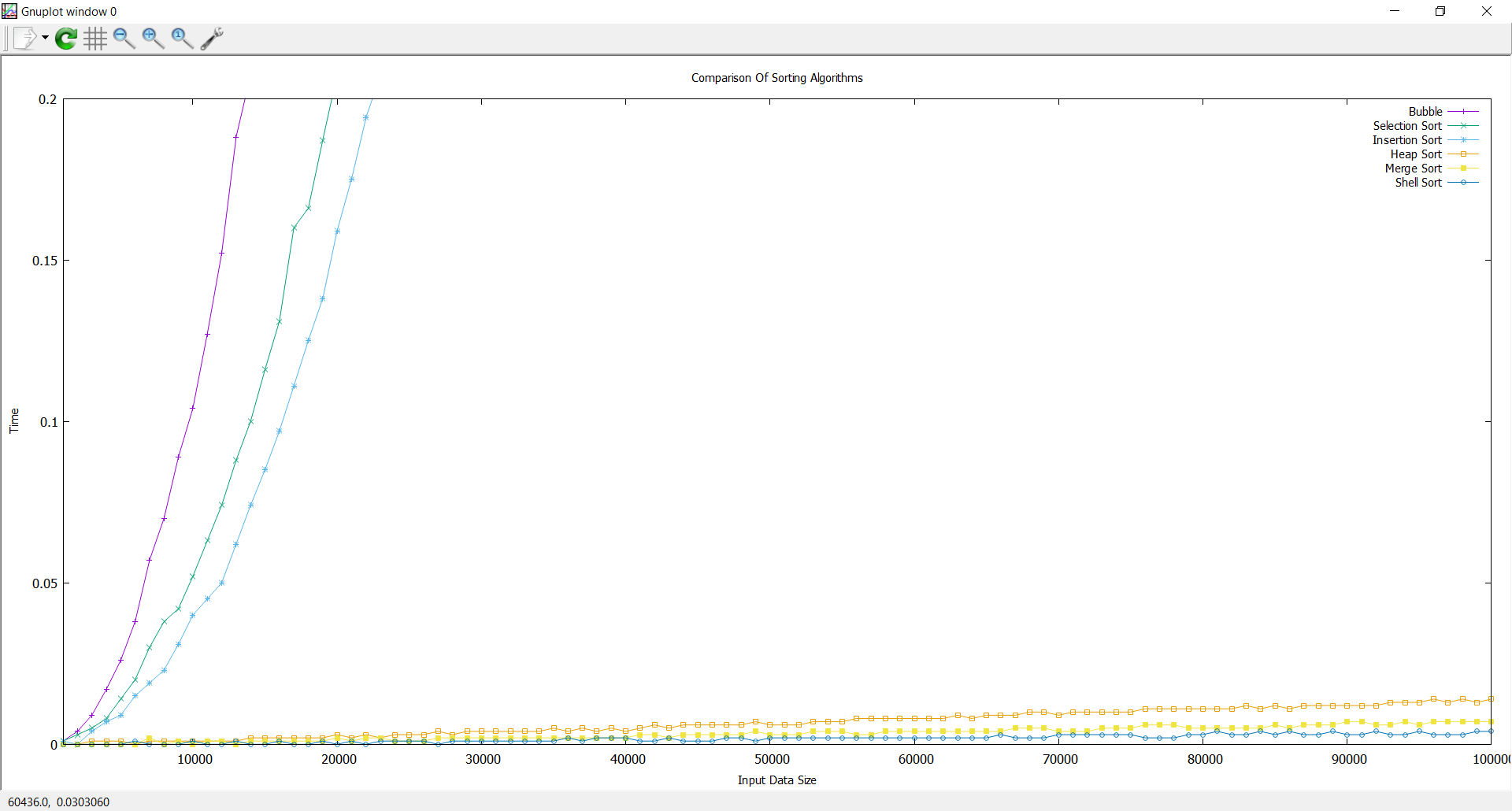
    fclose(fptr3);

    fclose(fptr4);

    fclose(fptr5);

    fclose(fptr6);

}

Graph:

Best Sorting algorithm :

In my opinion, Heap Sort is the best sorting algorithm as it works in O(nlogn) always and doesn’t consume much extra space.