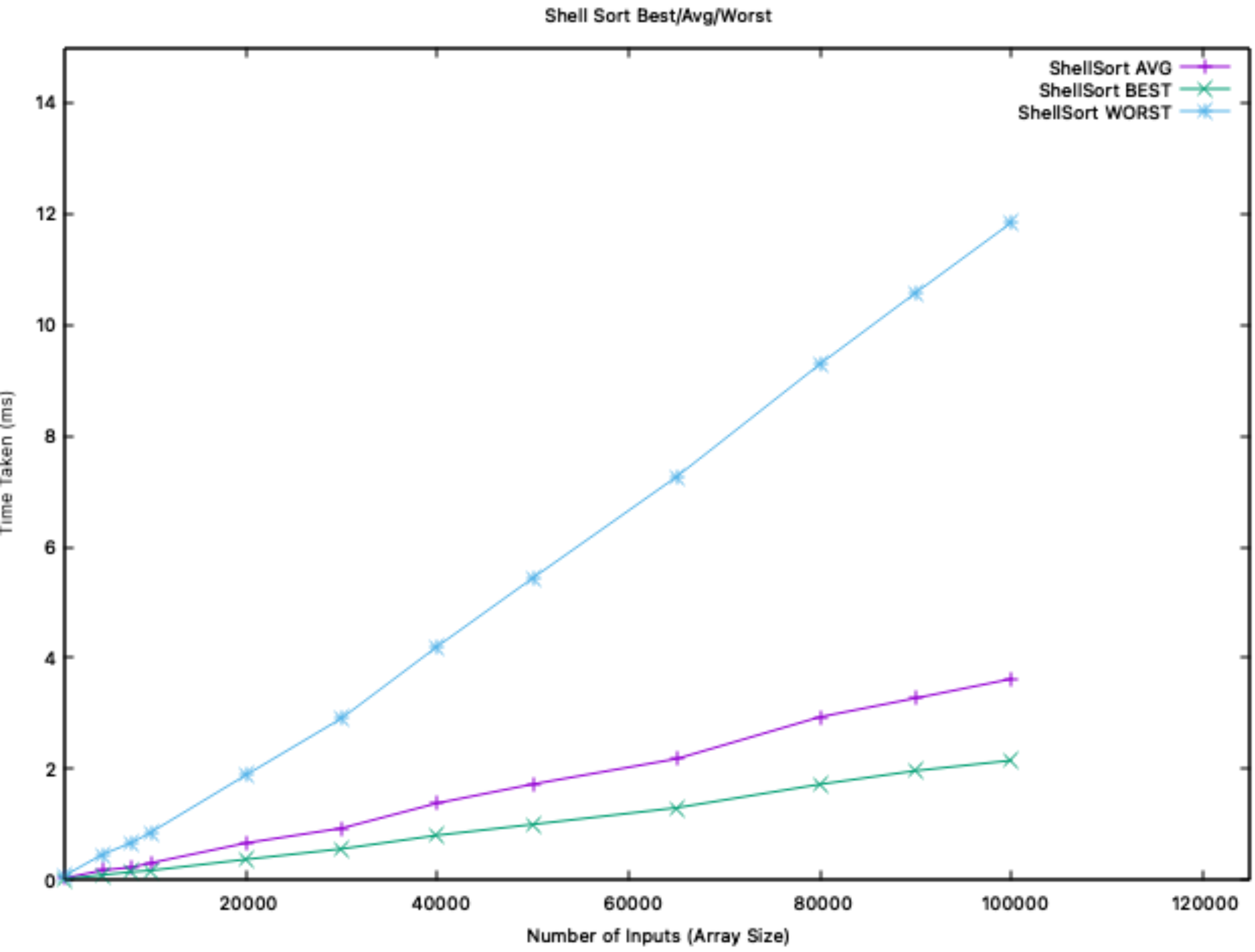
NAME:- Sourav Paul
GROUP:- D2
REG. NO:- 20214056
BRANCH:- CSE DEPT.

ASSIGNMENT - 03

1). Write a C Program to analyse the complexity of Shell Sort Algorithm. Also plot its graph for all cases.

Ans:- code

```
#include <stdio.h>

#include <stdlib.h>

#include <time.h>


clock_t begin;

clock_t end;

int arr[100000];

int size[] = {1000, 5000, 8000, 10000, 20000, 30000, 40000, 50000,
65000, 80000, 90000, 100000};


void swp(int *x, int *y) {
    int t = *x;
    *x = *y;
    *y = t;
}

void shSort(int n)
{
    int i, gap;
    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 writeTable(char* filename, int size, double time) {
    int i;
    FILE *fp = fopen(filename, "a+");

    if (fp == NULL) printf("FILE CANNOT BE OPENED\n");
    else {
        fprintf(fp, "%d %lf", size, time);
        fprintf(fp, "\n");
    }
    fclose(fp);
}

void readData(char* filename) {
    FILE *fp = fopen(filename, "r+");
    char x[16];
    int i, k = 0;

    if (fp == NULL) printf("FILE CANNOT BE OPENED\n");
    else {
        while (fgets(x, 16, fp) != NULL) {
            int num = 0;
            fscanf(fp, "%d", &num);
            if (num == 0) break;
            arr[k++] = num;
        }
    }
    fclose(fp);
}

int main(int argc, char** argv) {
    int i;
    for (i = 0; i <= 11; i++) {
        readData(argv[1]);
        begin = clock();
        shSort(size[i]);
        end = clock();
        writeTable(argv[2], size[i], (end - begin) / 2000.0);
    }
    return 0;
}
```



2). 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

Ans:- code

```
// Analysis of Selection Sort over 100K entries

#include <stdio.h>
#include <stdlib.h>
#include <time.h>

clock_t begin;
clock_t end;
int arr[100000];
int size[] = {1000, 5000, 8000, 10000, 20000, 30000, 40000, 50000,
65000, 80000, 90000, 100000};

void swp(int *x, int *y) {
   int t = *x;
```

```
    *x = *y;
    *y = t;
}

void sSort(int n) {
    int i, j, min;
    for (i = 0; i < n - 1; i++) {
        min = i;
        for (j = i; j < n; j++) {
            if (arr[j] < arr[min])
                min = j;
        }
        swp(&arr[min], &arr[i]);
    }
}

void writeTable(char* filename, int size, double time) {
    int i;
    FILE *fp = fopen(filename, "a+");

    if (fp == NULL) printf("FILE CANNOT BE OPENED\n");
    else {
        fprintf(fp, "%d %lf", size, time);
        fprintf(fp, "\n");
    }
    fclose(fp);
}

void readData(char* filename) {
    FILE *fp = fopen(filename, "r+");
    char x[16];
    int i, k = 0;

    if (fp == NULL) printf("FILE CANNOT BE OPENED\n");
    else {
        while (fgets(x, 16, fp) != NULL) {
            int num = 0;
            fscanf(fp, "%d", &num);
            if (num == 0) break;
            arr[k++] = num;
        }
    }
    fclose(fp);
}

int main(int argc, char** argv) {
   for (int i = 0; i <= 11; i++) {
             readData(argv[1]);
      begin = clock();
      sSort(size[i]);
      end = clock();
      writeTable(argv[2], size[i], (end - begin) / 2000.0);
   }
   return 0;
}

// Analysis of BubbleSort over 100K entries

#include <stdio.h>
#include <stdlib.h>
#include <time.h>

clock_t begin;
clock_t end;
int arr[100000];
int size[] = {1000, 5000, 8000, 10000, 20000, 30000, 40000, 50000,
65000, 80000, 90000, 100000};

void writeRand()
```

```
{
   int i, toWrite=0;
       int size = 100000;
       FILE *fp =  fopen("best.txt", "w+");
   if(fp == NULL) printf("FILE CANNOT BE OPENED\n");
       else
       {
       fprintf(fp, "%d", size);
      fprintf(fp, "\n");
      for(i=1; i<=100000; i++)
      {
         toWrite = arr[i];
         fprintf(fp, "%d", toWrite);
         fprintf(fp, "\n");
      }
   }
       fclose(fp);
}

void swp(int *x, int *y) {
   int t = *x;
   *x = *y;
   *y = t;
}

void bSort(int n) {
    int i, j, t;
    for (i = 0; i < n; i++) {
        for (j = 0; j < n - i - 1; j++) {
            if (arr[j + 1] < arr[j])
                swp(&arr[j], &arr[j + 1]);
        }
    }
}

void writeTable(char* filename, int size, double time) {
    int i;
    FILE *fp = fopen(filename, "a+");

    if (fp == NULL) printf("FILE CANNOT BE OPENED\n");
    else {
        fprintf(fp, "%d %lf", size, time);
        fprintf(fp, "\n");
    }
    fclose(fp);
}
```

```
void readData(char* filename) {
    FILE *fp = fopen(filename, "r+");
    char x[16];
    int i, k = 0;

    if (fp == NULL) printf("FILE CANNOT BE OPENED\n");
    else {
        while (fgets(x, 16, fp) != NULL) {
            int num = 0;
            fscanf(fp, "%d", &num);
            if (num == 0) break;
            arr[k++] = num;
        }
    }
    fclose(fp);
}

int main(int argc, char** argv) {
    for (int i = 0; i <= 11; i++) {
                readData(argv[1]);
        begin = clock();
        bSort(size[i]);
        end = clock();
        writeTable(argv[2], size[i], (end - begin) / 2000.0);
    }
    return 0;
}

// Analysis of Insertion Sort over 100K entries

#include <stdio.h>
#include <stdlib.h>
#include <time.h>

clock_t begin;
clock_t end;
int arr[100000];
int size[] = {1000, 5000, 8000, 10000, 20000, 30000, 40000, 50000,
65000, 80000, 90000, 100000};

void iSort(int n) {
    int i, j, curr, t;
    for (i = 1; i < n; i++) {
        curr = arr[i];
        t = i;
        while (t > 0 && arr[t - 1] > curr) {
            arr[t] = arr[t - 1];
            t = t - 1;
        }
        arr[t] = curr;
    }
}

void writeTable(char* filename, int size, double time) {
    int i;
    FILE *fp = fopen(filename, "a+");

    if (fp == NULL) printf("FILE CANNOT BE OPENED\n");
    else {
        fprintf(fp, "%d %lf", size, time);
        fprintf(fp, "\n");
    }
    fclose(fp);
}

void readData(char* filename) {
    FILE *fp = fopen(filename, "r+");
    char x[16];
    int i, k = 0;

    if (fp == NULL) printf("FILE CANNOT BE OPENED\n");
    else {
        while (fgets(x, 16, fp) != NULL) {
            int num = 0;
            fscanf(fp, "%d", &num);
            if (num == 0) break;
            arr[k++] = num;
        }
    }
    fclose(fp);
}

int main(int argc, char** argv) {
    for (int i = 0; i <= 11; i++) {
                readData(argv[1]);
        begin = clock();
        iSort(size[i]);
        end = clock();
        writeTable(argv[2], size[i], (end - begin) / 2000.0);
    }
    return 0;
```

```
}

// Analysis of Merge Sort over 100K entries

#include <stdio.h>
#include <stdlib.h>
#include <time.h>

clock_t begin;
clock_t end;

void swap(int *x, int *y) {
   int temp = *x;
   *x = *y;
   *y = temp;
}

void merge(int A[],int l,int mid,int r)
{
   int i=l,j=mid+1,k=l;
   int B[100000];
   while(i<=mid && j<=r)
   {
        if(A[i]<A[j])
            B[k++]=A[i++];
        else
            B[k++]=A[j++];
    }
    for(;i<=mid;i++)
        B[k++]=A[i];
    for(;j<=r;j++)
        B[k++]=A[j];
    for(i=l;i<=r;i++)
        A[i]=B[i];
}

void mSort(int A[],int l,int r)
{
    int mid;
    if(l<r)
    {
        mid=(l+r)/2;
        mSort(A,l,mid);
        mSort(A,mid+1,r);
        merge(A,l,mid,r);
    }
```

```
}

void writeTable(int size, double time, char *filename) {
   int i;
   FILE *fp = fopen(filename, "a+");

   if (fp == NULL) printf("FILE CANNOT BE OPENED\n");
   else {
      fprintf(fp, "%d %lf", size, time);
      fprintf(fp, "\n");
   }
   fclose(fp);
}

void readData(int arr[], char *filename) {
   FILE *fp = fopen(filename, "r+");
   char x[16];
   int i, k = 0;

   if (fp == NULL) printf("FILE CANNOT BE OPENED\n");
   else {
      while (fgets(x, 16, fp) != NULL) {
         int num = 0;
```

```
            fscanf(fp, "%d", &num);
            if (num == 0) break;
            arr[k++] = num;
        }
    }
    fclose(fp);
}

int main(int argc, char **argv) {
    int arr[100000];
    int size[] = {1000, 5000, 8000, 10000, 20000, 30000, 40000,
50000, 65000, 80000, 90000, 100000};
    int i = 0;
    for (i = 0; i <= 11; i++) {
        readData(arr, argv[1]);
        begin = clock();
        mSort(arr, 0, size[i]);
        end = clock();
        writeTable(size[i], (end - begin) / 2000, argv[2]);
    }
    return 0;
}
```

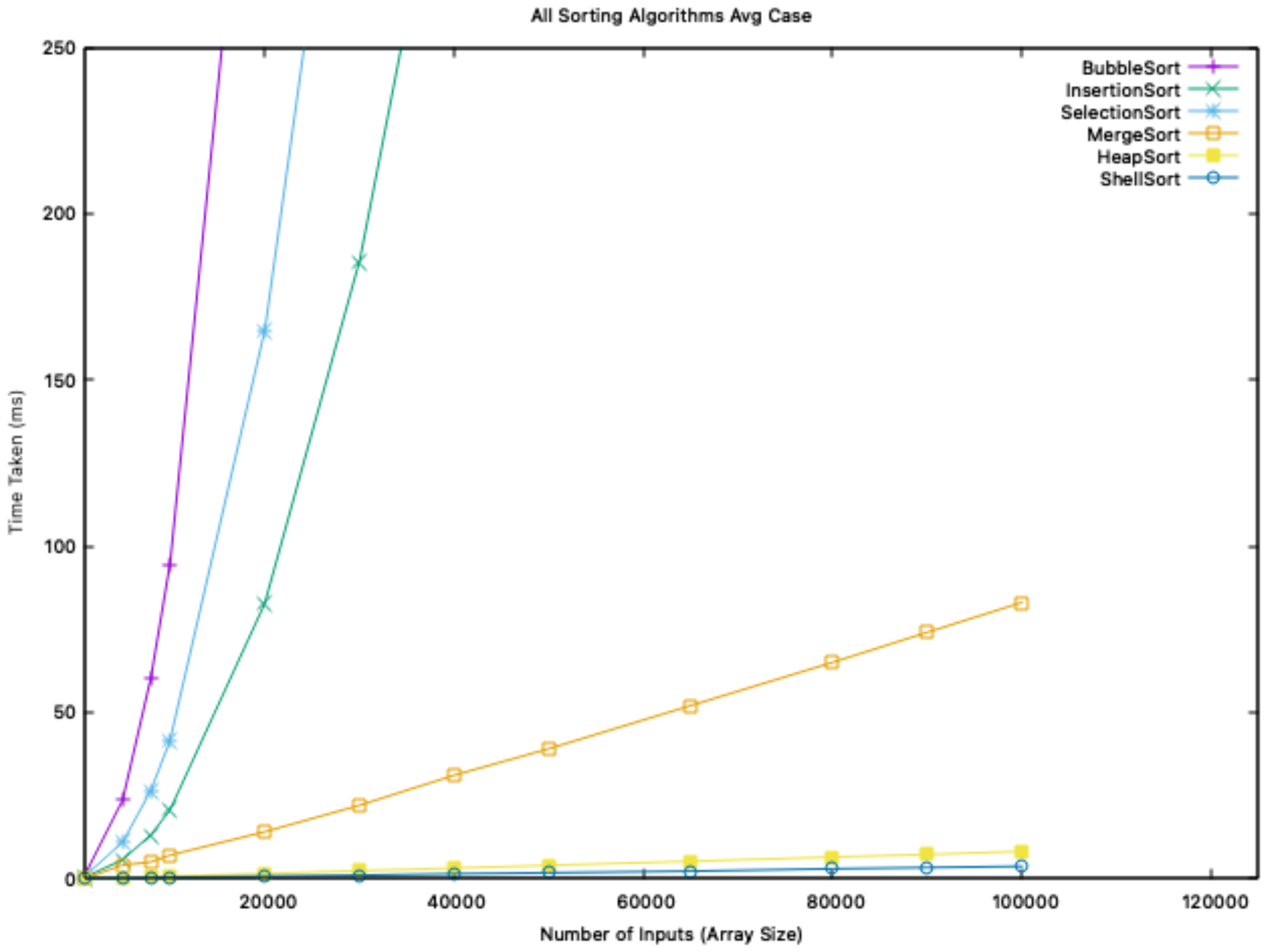
```
}

// Analysis of Heap Sort over 100K entries

#include <stdio.h>
#include <stdlib.h>
#include <time.h>

clock_t begin;
clock_t end;

void insertMaxHeap(int arr[], int n)
{
        int t, i=n;
        t=arr[n];

        while(i>1 && t>arr[i/2])
        {
                arr[i] = arr[i/2];
                i/=2;
        }
        arr[i] = t;
}

int removeMaxHeap(int arr[], int n)
```

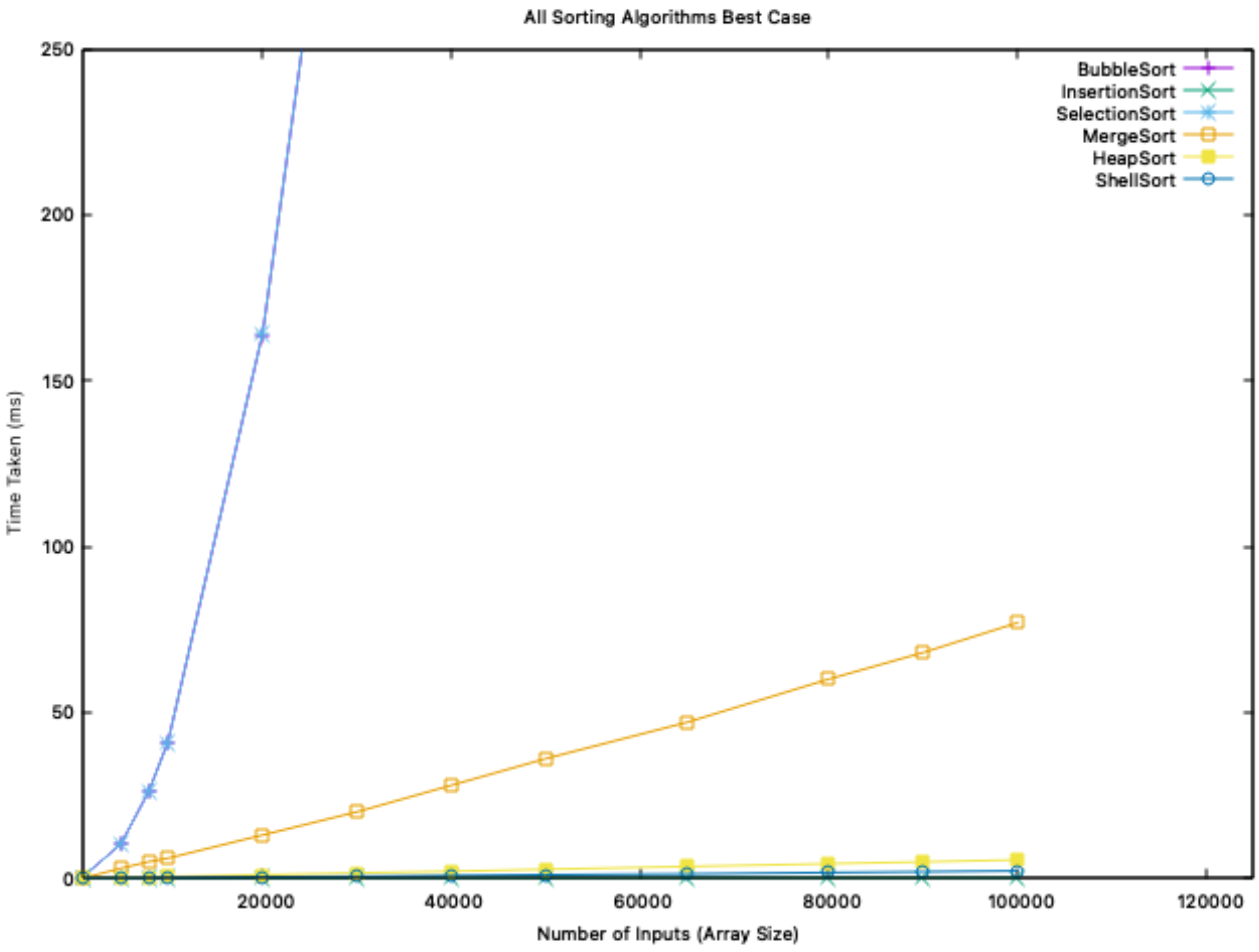
```
{
        int i, j, x, t, val;
        val=arr[1];
        x=arr[n];
        arr[1]=arr[n];
        arr[n]=val;
        i=1; j=i*2;

        while(j <= n-1)
        {
                if(j<n-1 && arr[j+1] > arr[j])
                        j++;
                if(arr[i] < arr[j])
                {
                        t=arr[i];
                        arr[i] = arr[j];
                        arr[j] = t;
                        i=j;
                        j*=2;
                }
                else break;
        }
        return val;
```

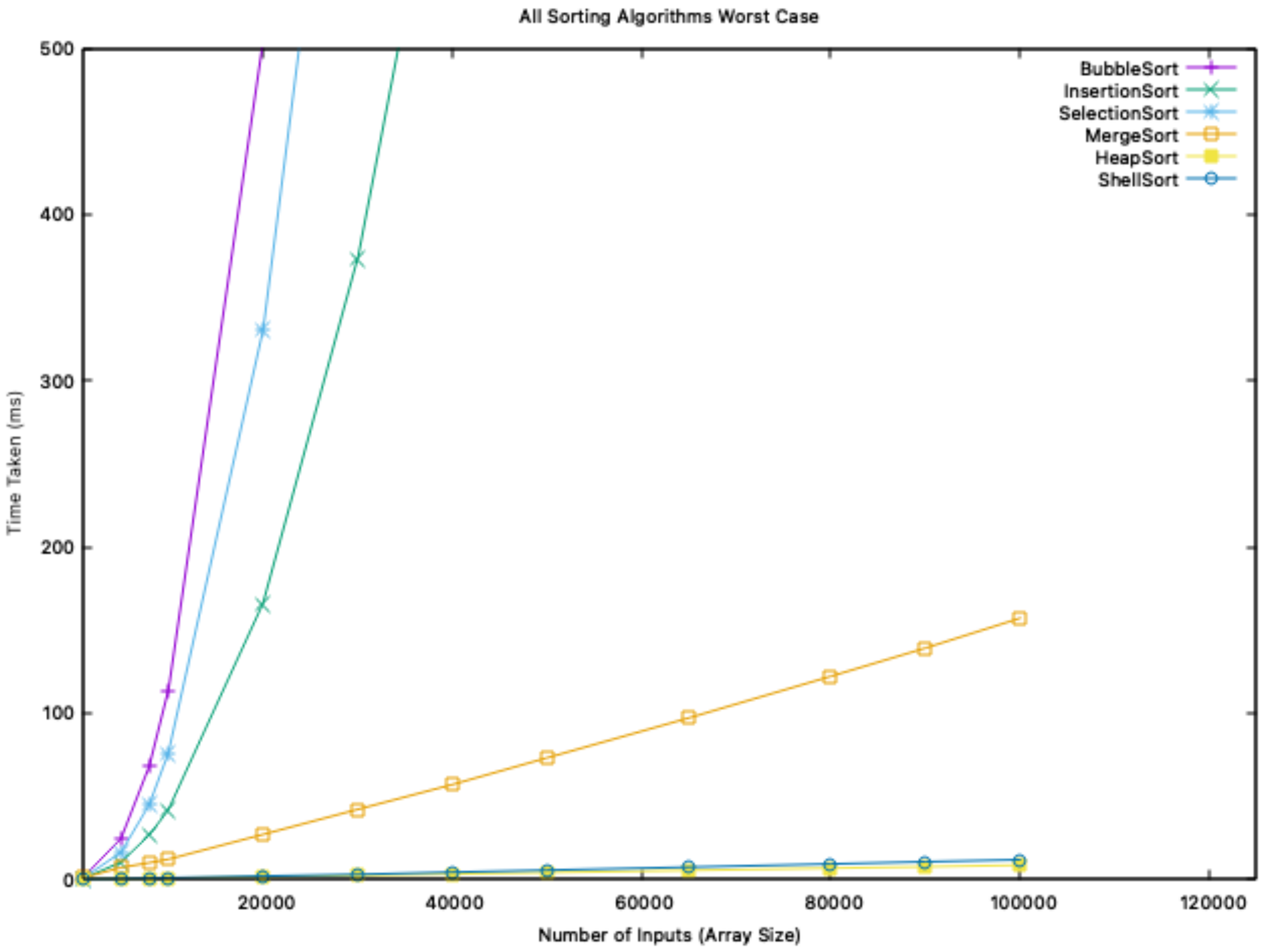
```
}

void createMaxHeap(int arr[], int n)
{
        int i;
        for(i=2; i<n; i++)
                insertMaxHeap(arr, i);
}

void hSort(int arr[], int n)
{
        int i;
        for(i=2; i<n; i++)
                insertMaxHeap(arr, i);
        for(i=n-1; i>1; i--)
                removeMaxHeap(arr, i);
}

void writeTable(char* filename, int size, double time) {
    int i;
    FILE *fp = fopen(filename, "a+");

    if (fp == NULL) printf("FILE CANNOT BE OPENED\n");
    else {
        fprintf(fp, "%d %lf", size, time);
        fprintf(fp, "\n");
    }
    fclose(fp);
}

void readData(int arr[], char* filename) {
    FILE *fp = fopen(filename, "r+");
    char x[16];
    int i, k = 0;

    if (fp == NULL) printf("FILE CANNOT BE OPENED\n");
    else {
        while (fgets(x, 16, fp) != NULL) {
            int num = 0;
            fscanf(fp, "%d", &num);
            if (num == 0) break;
            arr[k++] = num;
        }
    }
    fclose(fp);
}
```

```
int main(int argc, char** argv) {
        int arr[100000];
        int size[] = {1000, 5000, 8000, 10000, 20000, 30000, 40000,
50000, 65000, 80000, 90000, 100000};
    int i;
        for (int i = 0; i <= 11; i++) {
                readData(arr, argv[1]);
        begin = clock();
        hSort(arr, size[i]);
        end = clock();
        writeTable(argv[2], size[i], (end - begin) / 2000.0);
    }
    return 0;
}


// Analysis of ShellSort over 100K entries


#include <stdio.h>
#include <stdlib.h>
#include <time.h>


clock_t begin;
clock_t end;

int arr[100000];

int size[] = {1000, 5000, 8000, 10000, 20000, 30000, 40000, 50000,
65000, 80000, 90000, 100000};


void swp(int *x, int *y) {

   int t = *x;

   *x = *y;

   *y = t;

}


void shSort(int n)

{

      int i, gap;

   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 writeTable(char* filename, int size, double time) {
    int i;
    FILE *fp = fopen(filename, "a+");

    if (fp == NULL) printf("FILE CANNOT BE OPENED\n");
    else {
        fprintf(fp, "%d %lf", size, time);
        fprintf(fp, "\n");
    }
    fclose(fp);
}

void readData(char* filename) {
    FILE *fp = fopen(filename, "r+");
    char x[16];
    int i, k = 0;

    if (fp == NULL) printf("FILE CANNOT BE OPENED\n");
    else {
        while (fgets(x, 16, fp) != NULL) {
            int num = 0;
            fscanf(fp, "%d", &num);
            if (num == 0) break;
            arr[k++] = num;
        }
    }
    fclose(fp);
}

int main(int argc, char** argv) {
    int i;
    for (i = 0; i <= 11; i++) {
                readData(argv[1]);
                begin = clock();
        shSort(size[i]);
        end = clock();
        writeTable(argv[2], size[i], (end - begin) / 2000.0);
    }
    return 0;
}
```


All Sorting Algorithms Avg Case
BubbleSort
InsertionSort
SelectionSort
MergeSort
HeapSort
ShellSort
Time Taken (ms)
Number of Inputs (Array Size)
0
50
100
150
200
250
20000
40000
60000
80000
100000
120000


All Sorting Algorithms Best Case
Time Taken (ms)
Number of Inputs (Array Size)
BubbleSort
InsertionSort
SelectionSort
MergeSort
HeapSort
ShellSort
0
50
100
150
200
250
20000
40000
60000
80000
100000
120000


All Sorting Algorithms Worst Case
Time Taken (ms)
Number of Inputs (Array Size)
0
100
200
300
400
500
20000
40000
60000
80000
100000
120000
BubbleSort
InsertionSort
SelectionSort
MergeSort
HeapSort
ShellSort