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

ASSIGNMENT - 05

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

Ans:-

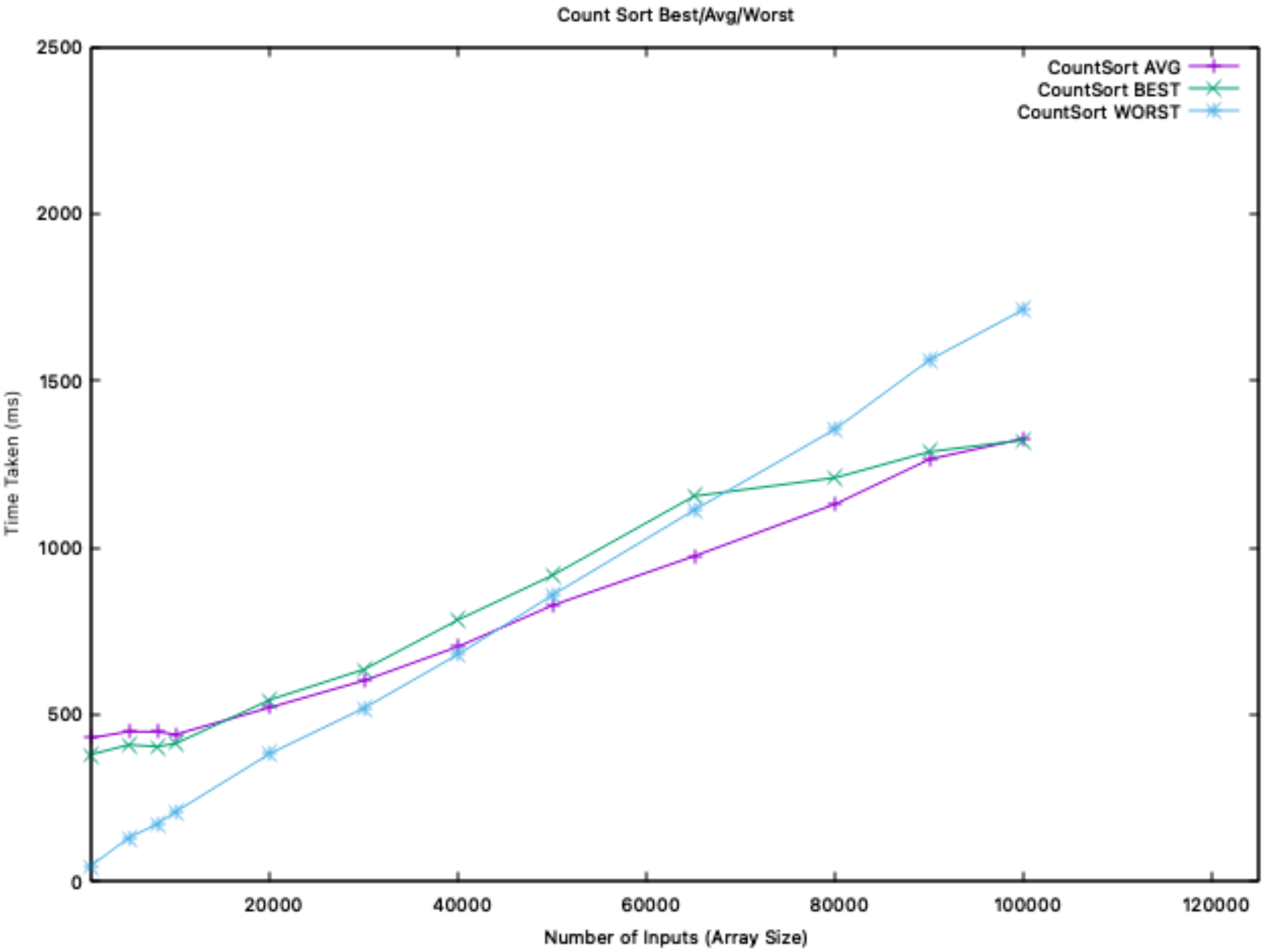
```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

clock_t begin;
clock_t end;

int findMax(int arr[], int n) {
   int i, mx = -16777216;
   for (i = 0; i < n; i++) {
      if (arr[i] > mx)
          mx = arr[i];
   }
   return mx;
}

void cSort(int arr[], int n) {
   int mx = findMax(arr, n);
```

```
    int i, j, *c;
    c = (int *) malloc(sizeof(int) * mx + 1);

    for (i = 0; i < mx + 1; i++)
        c[i] = 0;
    for (i = 0; i < n; i++)
        c[arr[i]]++;

    i = 0;
    j = 0;
    while (j < mx + 1) {
        if (c[j] > 0) {
            arr[i++] = j;
            c[j]--;
        } else j++;
    }
}

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();
        cSort(arr, size[i]);
        end = clock();
        writeTable(size[i], (end - begin), argv[2]);
    }
    return 0;
}
```

Dataplot.p for Counting Sort

```
set autoscale                    # scale axes automatically
unset log                        # remove any log-scaling
unset label                      # remove any previous labels
set xtic auto                    # set xtics automatically
set ytic auto                    # set ytics automatically        set tics
font "Helvetica,10"
```

```
set title "Count Sort Best/Avg/Worst"
set xlabel "Number of Inputs (Array Size)"
set ylabel "Time Taken (ms)"

#set key 0.01,100
#set label "Yield Point" at 0.003,260
#set arrow from 0.0028,250 to 0.003,280

set xr [1000:125000]
set yr [0:2500]
plot    "cTableAVG.txt" using 1:2 title 'CountSort AVG' with
linespoints, \
        "cTableBST.txt" using 1:2 title 'CountSort BEST' with
linespoints, \
        "cTableWST.txt" using 1:2 title 'CountSort WORST' with
linespoints
```



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

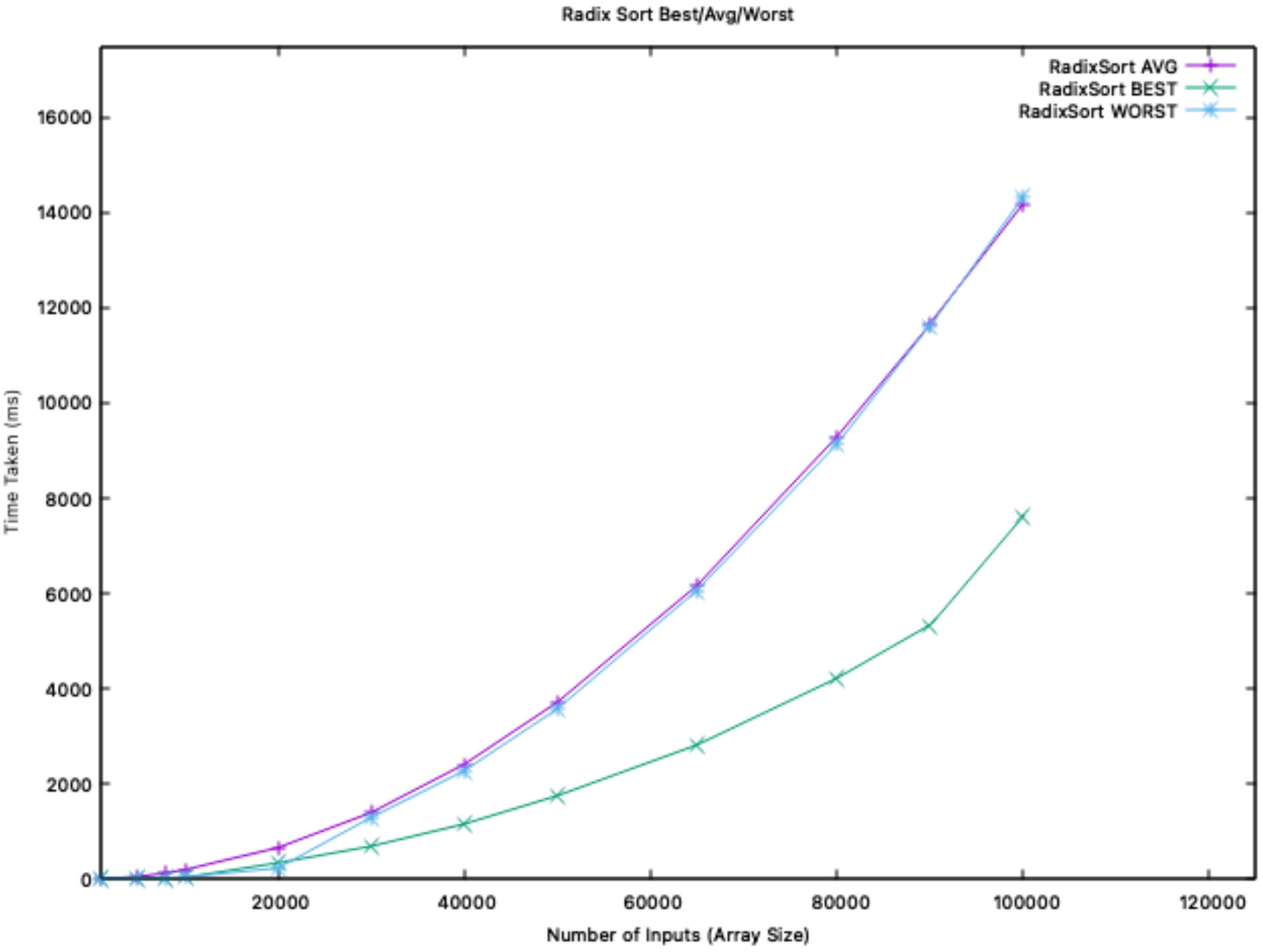
Ans:-

```
#include <iostream>
#include <stdlib.h>
#include <stdio.h>
#include <math.h>
#include <ctime>

using namespace std;

clock_t st;
clock_t en;

template<class T>
void Print(T &vec, int n, string s) {
   cout << s << ": [" << flush;
   for (int i = 0; i < n; i++) {
      cout << vec[i] << flush;
      if (i < n - 1) {
         cout << ", " << flush;
      }
   }
   cout << "]" << endl;
}

int Max(int A[], int n) {
    int max = -32768;
    for (int i = 0; i < n; i++) {
        if (A[i] > max) {
            max = A[i];
        }
    }
    return max;
}

class Node {
public:
    int value;
    Node *next;
};

int countDigits(int x) {
    int count = 0;
    while (x != 0) {
        x = x / 10;
        count++;
    }
    return count;
}

void initializeBins(Node **p, int n) {
    for (int i = 0; i < n; i++) {
        p[i] = nullptr;
    }
}

void Insert(Node **ptrBins, int value, int idx) {
    Node *temp = new Node;
    temp->value = value;
    temp->next = nullptr;

    if (ptrBins[idx] == nullptr) {
        ptrBins[idx] = temp;
    } else {
        Node *p = ptrBins[idx];
        while (p->next != nullptr) {
            p = p->next;
        }
        p->next = temp;
    }
}

int Delete(Node **ptrBins, int idx) {
    Node *p = ptrBins[idx];
    ptrBins[idx] = ptrBins[idx]->next;
    int x = p->value;
    delete p;
    return x;
}

int getBinIndex(int x, int idx) {
    return (int) (x / pow(10, idx)) % 10;
}

void rSort(int A[], int n) {
    int max = Max(A, n);
    int nPass = countDigits(max);
    Node **bins = new Node *[10];
    initializeBins(bins, 10);

    for (int pass = 0; pass < nPass; pass++) {
        for (int i = 0; i < n; i++) {
            int binIdx = getBinIndex(A[i], pass);
            Insert(bins, A[i], binIdx);
        }

        int i = 0;
        int j = 0;
    while (i < 10) {
      while (bins[i] != nullptr) {
        A[j++] = Delete(bins, i);
      }
      i++;
    }
    initializeBins(bins, 10);
  }
  delete[] bins;
}

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]);
        st = clock();
        rSort(arr, size[i]);
        en = clock();
        writeTable(size[i], (en - st) / 2000, argv[2]);
    }
    return 0;
}
```

Dataplot.p for Radix Sort

```
set autoscale                    # scale axes automatically
unset log                      # remove any log-scaling
unset label                     # remove any previous labels
set xtic auto                   # set xtics automatically
set ytic auto                   # set ytics automatically         set tics
font "Helvetica,10"
set title "Radix Sort Best/Avg/Worst"
set xlabel "Number of Inputs (Array Size)"
set ylabel "Time Taken (ms)"

#set key 0.01,100
#set label "Yield Point" at 0.003,260
```

```
#set arrow from 0.0028,250 to 0.003,280

set xr [1000:125000]
set yr [0:17500]
plot    "rTableAVG.txt" using 1:2 title 'RadixSort AVG' with
linespoints, \
        "rTableBST.txt" using 1:2 title 'RadixSort BEST' with
linespoints, \
        "rTableWST.txt" using 1:2 title 'RadixSort WORST' with
linespoints
```



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

Ans:-

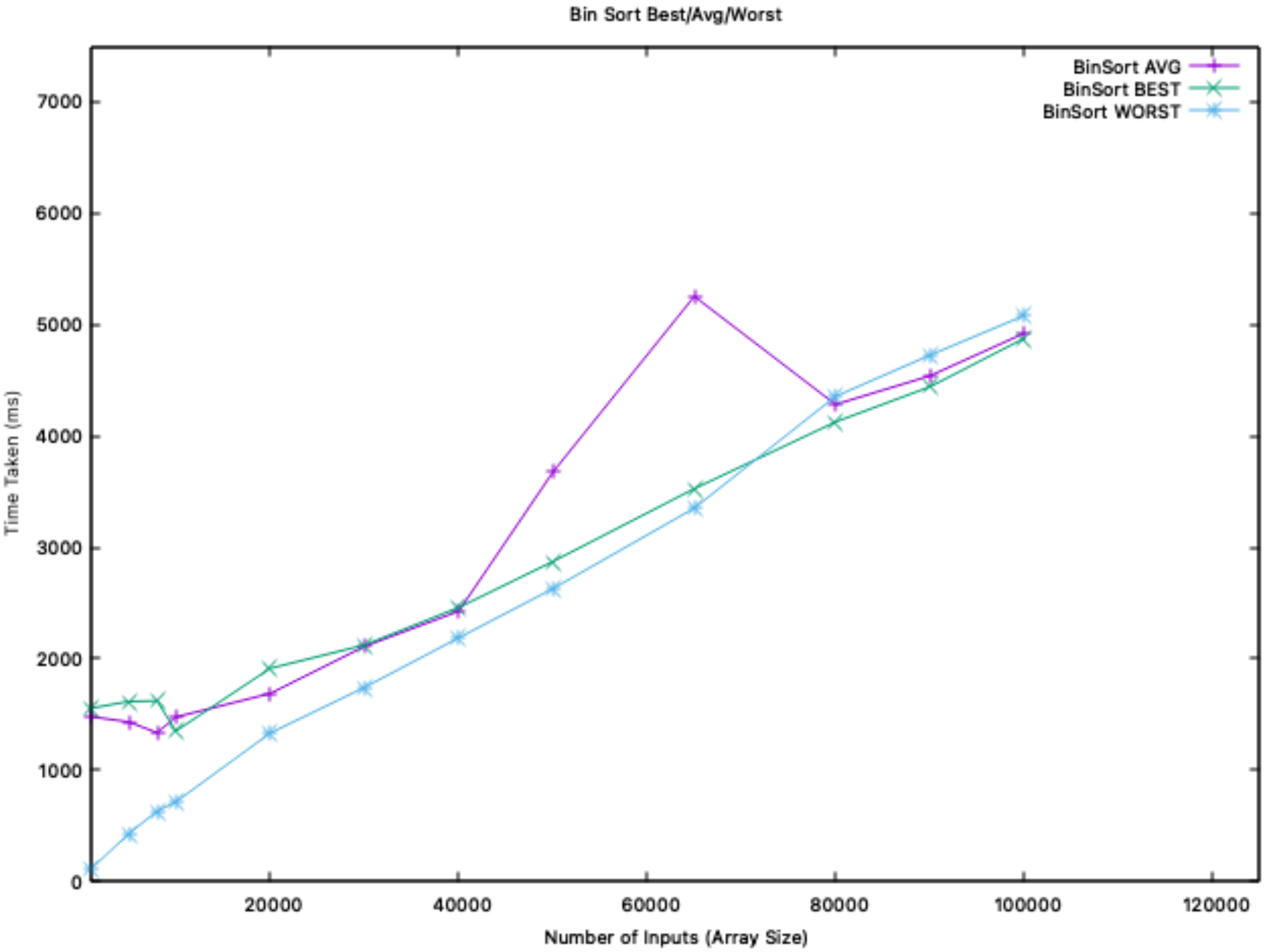
```
#include <iostream>
#include <stdlib.h>
#include <stdio.h>
#include <math.h>
#include <ctime>

using namespace std;

clock_t st;
clock_t en;

template<class T>
void Print(T &vec, int n, string s) {
   cout << s << ": [" << flush;
   for (int i = 0; i < n; i++) {
      cout << vec[i] << flush;
      if (i < n - 1) {
         cout << ", " << flush;
      }
   }
   cout << "]" << endl;
}

int Max(int A[], int n) {
   int max = -32768;
   for (int i = 0; i < n; i++) {
        if (A[i] > max) {
            max = A[i];
        }
    }
    return max;
}

class Node {
public:
    int value;
    Node *next;
};

void Insert(Node **ptrBins, int idx) {
    Node *temp = new Node;
    temp->value = idx;
    temp->next = nullptr;

    if (ptrBins[idx] == nullptr) {
        ptrBins[idx] = temp;
    } else {
        Node *p = ptrBins[idx];
        while (p->next != nullptr) {
            p = p->next;
        }
        p->next = temp;
    }
}

int Delete(Node **ptrBins, int idx) {
    Node *p = ptrBins[idx];
    ptrBins[idx] = ptrBins[idx]->next;
    int x = p->value;
    delete p;
    return x;
}
```

```
void bucketSort(int A[], int n) {
   int max = Max(A, n);
   Node **bins = new Node *[max + 1];

   for (int i = 0; i < max + 1; i++) {
      bins[i] = nullptr;
   }

   for (int i = 0; i < n; i++) {
      Insert(bins, A[i]);
   }

   int i = 0;
   int j = 0;
   while (i < max + 1) {
      while (bins[i] != nullptr) {
         A[j++] = Delete(bins, i);
      }
      i++;
   }
   delete[] bins;
}

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]);
      st = clock();
      bucketSort(arr, size[i]);
      en = clock();
      writeTable(size[i], (en - st), argv[2]);
   }
   return 0;
}
```

Dataplot.p for Bucket Sort

```
set autoscale                    # scale axes automatically
unset log                       # remove any log-scaling
unset label                      # remove any previous labels
set xtic auto                    # set xtics automatically
set ytic auto                    # set ytics automatically        set tics
font "Helvetica,10"
set title "Bin Sort Best/Avg/Worst"
set xlabel "Number of Inputs (Array Size)"
set ylabel "Time Taken (ms)"

#set key 0.01,100
#set label "Yield Point" at 0.003,260
#set arrow from 0.0028,250 to 0.003,280

set xr [1000:125000]
set yr [0:7500]
plot    "bnTableAVG.txt" using 1:2 title 'BinSort AVG' with
linespoints, \
       "bnTableBST.txt" using 1:2 title 'BinSort BEST' with
linespoints, \
         "bnTableWST.txt" using 1:2 title 'BinSort WORST' with
linespoints
```


Bin Sort Best/Avg/Worst
BinSort AVG
BinSort BEST
BinSort WORST
Time Taken (ms)
0
1000
2000
3000
4000
5000
6000
7000
20000
40000
60000
80000
100000
120000
Number of Inputs (Array Size)