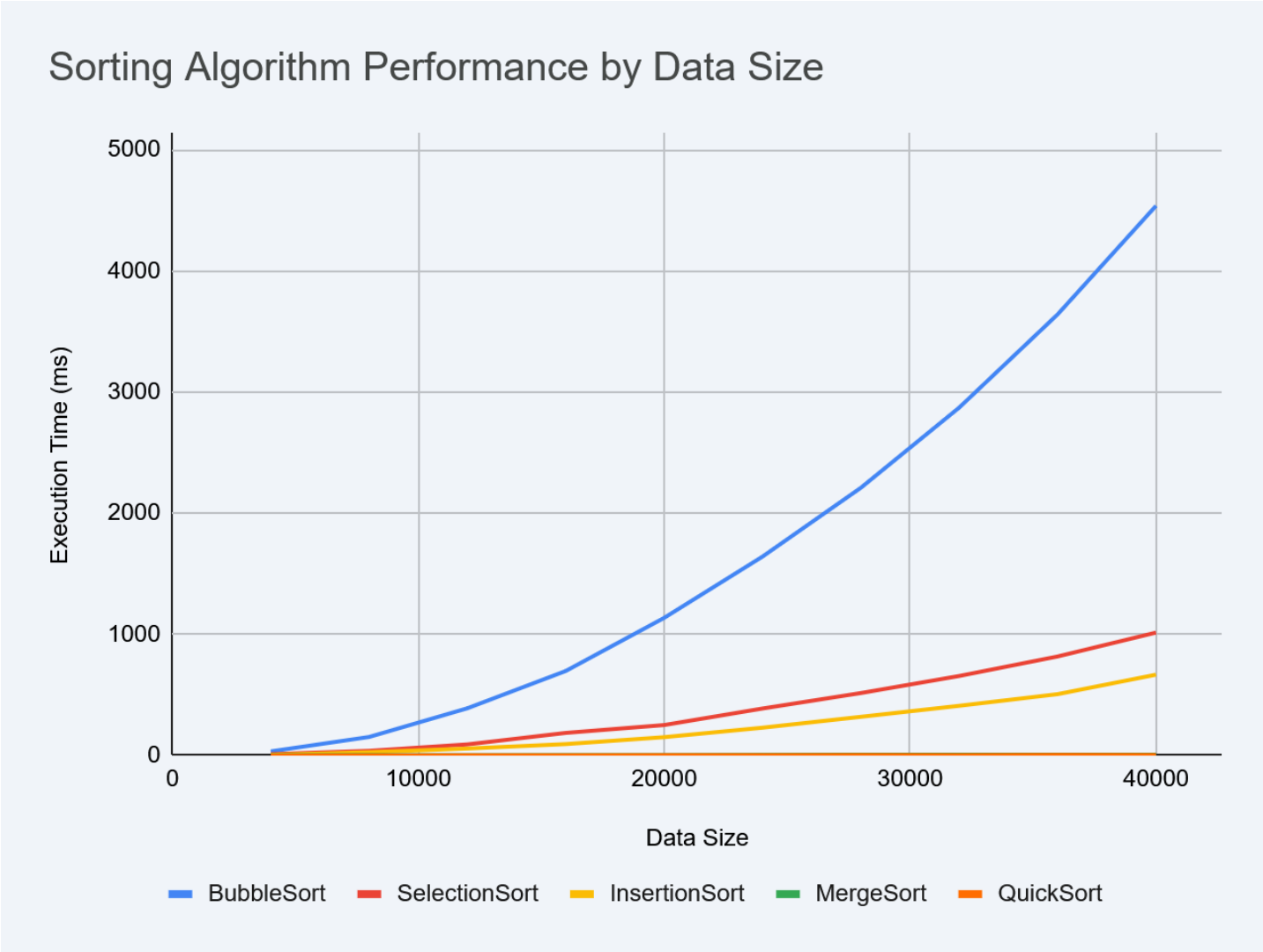
Algorithms Laboratory

Assignment-1

Deepak Maurya (23053044)

Graph:



Source Code:

```
#include "time.h"
#include "stdio.h"
#include "stdlib.h"
#include "string.h"

void swap(int *a, int *b) {
  int temp = *a;
  *a = *b;
  *b = temp;
}

void bubbleSort(int *arr, int n) {
  for (int i=0; i<n; i++) {
    for (int j=i; j<n; j++) {
      if (arr[i] > arr[j]) swap(&arr[i], &arr[j]);
    }
  }
}

void selectionSort(int *arr, int n) {
  for (int i=0; i<n; i++) {
    int k = i;
    for (int j=i; j<n; j++)
      if (arr[j] < arr[k]) k = j;

    if (k ≠ i) swap(&arr[i], &arr[k]);
  }
}

int partition(int *arr, int low, int high) {
  int pivot = arr[high];
  int i = (low - 1);

  for (int j = low; j ⩽ high - 1; j++) {
    if (arr[j] < pivot) {
      i++;
      swap(&arr[i], &arr[j]);
    }
  }
  swap(&arr[i + 1], &arr[high]);
  return (i + 1);
}

void quickSortRecursive(int *arr, int low, int high) {
  if (low < high) {
```

```
    int pi = partition(arr, low, high);

    quickSortRecursive(arr, low, pi - 1);
    quickSortRecursive(arr, pi + 1, high);
  }
}

void insertionSort(int *arr, int n) {
  for (int i = 1; i < n; i++) {
    int key = arr[i];
    int j = i - 1;

    while (j ⩾ 0 && arr[j] > key) {
      arr[j + 1] = arr[j];
      j = j - 1;
    }
    arr[j + 1] = key;
  }
}

void quickSort(int *arr, int n) {
  quickSortRecursive(arr, 0, n - 1);
}

void merge(int *arr, int left, int mid, int right) {
  int i, j, k;
  int n1 = mid - left + 1;
  int n2 = right - mid;

  int leftArr[n1], rightArr[n2];

  for (i = 0; i < n1; i++) leftArr[i] = arr[left + i];
  for (j = 0; j < n2; j++) rightArr[j] = arr[mid + 1 + j];

  i = 0; j = 0; k = left;
  while (i < n1 && j < n2) {
    if (leftArr[i] ⩽ rightArr[j]) arr[k++] = leftArr[i++];
    else arr[k++] = rightArr[j++];
  }

  while (i < n1) arr[k++] = leftArr[i++];
  while (j < n2) arr[k++] = rightArr[j++];
}

void mergeSortRecursive(int *arr, int left, int right) {
  if (left < right) {
    int mid = left + (right - left) / 2;
```

```
    mergeSortRecursive(arr, left, mid);
    mergeSortRecursive(arr, mid + 1, right);

    merge(arr, left, mid, right);
  }
}

void mergeSort(int *arr, int n) {
  mergeSortRecursive(arr, 0, n - 1);
}

int main() {
  srand(time(NULL));

  const int noOfBreakPoints = 10;
  int sizes[] = { 4000, 8000, 12000, 16000, 20000, 24000, 28000, 32000, 36000, 40000 };

  printf("Size,BubbleSort,SelectionSort,InsertionSort,MergeSort,QuickSort\n");

  for (int i = 0; i < noOfBreakPoints; i++) {
    int n = sizes[i];

    int *arr = (int*)malloc(n * sizeof(int));
    int *temp_arr = (int*)malloc(n * sizeof(int));

    if (arr == NULL || temp_arr == NULL) {
      printf("Memory allocation failed\n");
      return 1;
    }

    for (int j = 0; j < n; j++) arr[j] = rand() % INT_MAX;

    printf("%d,", n);

    clock_t start, end;
    double cpu_time_used_ms;

    memcpy(temp_arr, arr, n * sizeof(int));
    start = clock();
    bubbleSort(temp_arr, n);
    end = clock();
    cpu_time_used_ms = ((double)(end - start) * (double)1000) / CLOCKS_PER_SEC;
    printf("%.2f,", cpu_time_used_ms);

    memcpy(temp_arr, arr, n * sizeof(int));
    start = clock();
```

```
    selectionSort(temp_arr, n);
    end = clock();
    cpu_time_used_ms = ((double)(end - start) * (double)1000) / CLOCKS_PER_SEC;
    printf("%.2f,", cpu_time_used_ms);

    memcpy(temp_arr, arr, n * sizeof(int));
    start = clock();
    insertionSort(temp_arr, n);
    end = clock();
    cpu_time_used_ms = ((double)(end - start) * (double)1000) / CLOCKS_PER_SEC;
    printf("%.2f,", cpu_time_used_ms);

    memcpy(temp_arr, arr, n * sizeof(int));
    start = clock();
    mergeSort(temp_arr, n);
    end = clock();
    cpu_time_used_ms = ((double)(end - start) * (double)1000) / CLOCKS_PER_SEC;
    printf("%.2f,", cpu_time_used_ms);

    memcpy(temp_arr, arr, n * sizeof(int));
    start = clock();
    quickSort(temp_arr, n);
    end = clock();
    cpu_time_used_ms = ((double)(end - start) * (double)1000) / CLOCKS_PER_SEC;
    printf("%.2f", cpu_time_used_ms);

    printf("\n");

    free(arr);
    free(temp_arr);
  }

  return 0;
}
```