Aayush Shah
19BCE245
28 July 2021

Design and Analysis of Algorithms

Practical 1

- **Iterative approach :**

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

void swap(int *xp, int *yp){
    int temp = *xp;
    *xp = *yp;
    *yp = temp;
}

void bubbleSortIterative(int arr[], int n){
    int i, j;
    for (i = 0; i < n-1; i++)
        for (j = 0; j < n-i-1; j++)
            if (arr[j] > arr[j+1])
                swap(&arr[j], &arr[j+1]);
}

void insertionSortIterative(int arr[], int n){
    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;
    }
```

```
}

void selectionSortIterative(int arr[], int n){
  int i, j, min_idx;
  for (i = 0; i < n-1; i++){
       min_idx = i;
       for (j = i+1; j < n; j++)
            if (arr[j] < arr[min_idx])
            min_idx = j;
       swap(&arr[min_idx], &arr[i]);
  }
}

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

int main() {
  int lower = 0;
  int upper = 100000;
  int count = 200000;
  int arrForBubbleSort[count];
  int arrForInsertionSort[count];
  int arrForSelectionSort[count];

  clock_t start, end;
  double cpu_time_used;

  srand(time(0));

  for (int i = 0; i < count; i++){
       int num = (rand() % (upper - lower + 1)) + lower;
       arrForBubbleSort[i] = num;
       arrForInsertionSort[i] = num;
       arrForSelectionSort[i] = num;
  }

//     int arr[] = {64, 34, 25, 12, 22, 11, 90};
```

```
    int n = sizeof(arrForBubbleSort)/sizeof(arrForBubbleSort[0]);           //as all array are same -> size of all the array will be same.


    //BUBBLE SORT
    start = clock();
    bubbleSortIterative(arrForBubbleSort, n);
    end = clock();
    cpu_time_used = ((double) (end - start)) / CLOCKS_PER_SEC;
    printf("Sorted array in %f seconds with BUBBLE sort: \n",cpu_time_used);
//     printArray(arrForBubbleSort, n);


    //INSERTION SORT
    start = clock();
    insertionSortIterative(arrForInsertionSort, n);
    end = clock();
    cpu_time_used = ((double) (end - start)) / CLOCKS_PER_SEC;

    printf("Sorted array in %f seconds with INSERTION sort: \n",cpu_time_used);
//     printArray(arrForInsertionSort, n);

    //SELECTION SORT
    start = clock();
    selectionSortIterative(arrForSelectionSort, n);
    end = clock();
    cpu_time_used = ((double) (end - start)) / CLOCKS_PER_SEC;

    printf("Sorted array in %f seconds with SELECTION sort: \n",cpu_time_used);
//     printArray(arrForSelectionSort, n);

    return 0;
}
```

- **Recursive approach :**

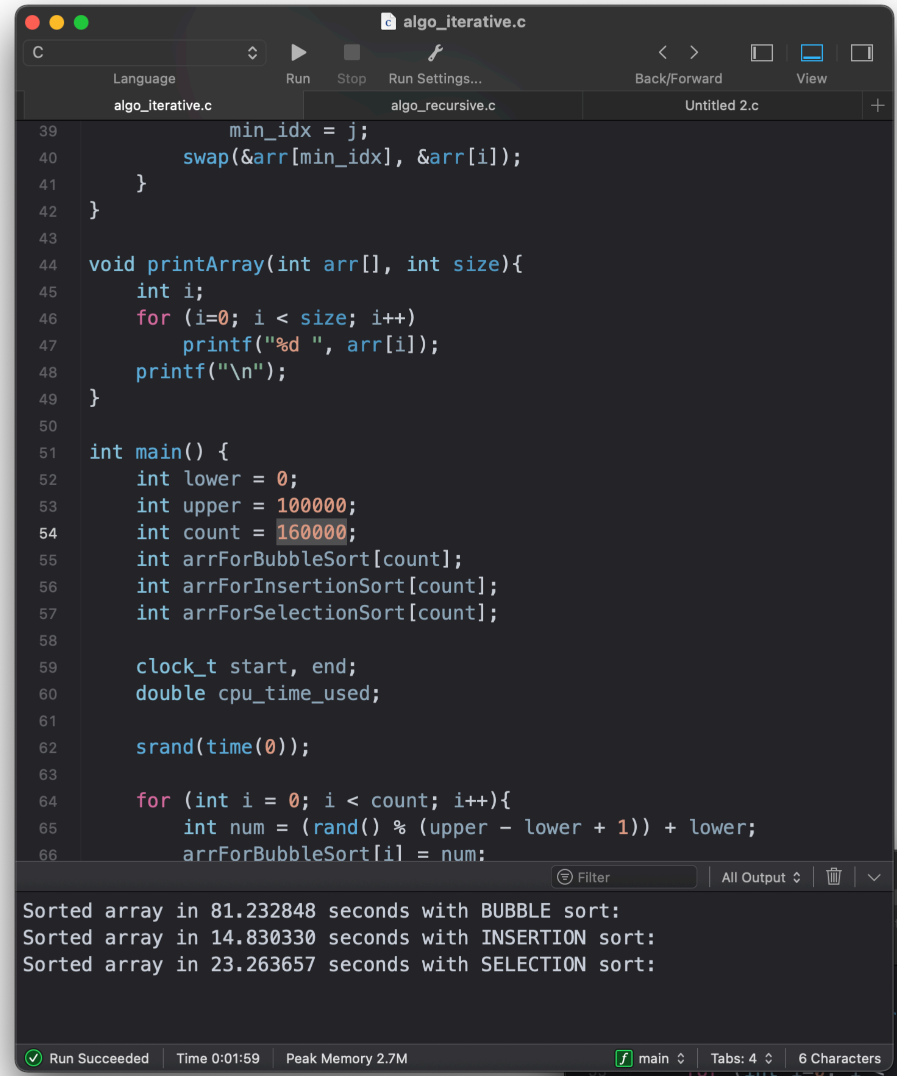
```c
#include <stdio.h>
#include <time.h>
#include <stdlib.h>

void swap(int *xp, int *yp){
  int temp = *xp;
  *xp = *yp;
  *yp = temp;
}

void bubbleSortRecursive(int arr[],int n){
  if(n==1)
       return;
  for(int i=0;i<n-1;i++)
       if(arr[i]>arr[i+1])
            swap(&arr[i],&arr[i+1]);
  bubbleSortRecursive(arr, n-1);
}

void insertionSortRecursive(int arr[], int n){
  if (n <= 1)
       return;
  insertionSortRecursive( arr, n-1 );
  int last = arr[n-1];
  int j = n-2;
  while (j >= 0 && arr[j] > last){
       arr[j+1] = arr[j];
       j--;
  }
  arr[j+1] = last;
}

int minIndex(int a[], int i, int j){
  if (i == j)
       return i;
  int k = minIndex(a, i + 1, j);
  return (a[i] < a[k])? i : k;
}

void selectionSortRecursive(int a[], int n, int index)
```

```
{
    if (index == n)
        return;
    int k = minIndex(a, index, n-1);
    if (k != index)
        swap(&a[k], &a[index]);

    selectionSortRecursive(a, n, index + 1);
}

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

int main() {
    int lower = 0;
    int upper = 100000;
    int count = 200000;
    int arrForBubbleSort[count];
    int arrForInsertionSort[count];
    int arrForSelectionSort[count];

    clock_t start, end;
    double cpu_time_used;

    srand(time(0));

    for (int i = 0; i < count; i++){
        int num = (rand() % (upper - lower + 1)) + lower;
        arrForBubbleSort[i] = num;
        arrForInsertionSort[i] = num;
        arrForSelectionSort[i] = num;
    }

    //   int arr[] = {64, 34, 25, 12, 22, 11, 90};
    int n = sizeof(arrForBubbleSort)/sizeof(arrForBubbleSort[0]);          //as all array are same -> size of all the array will be same.


//BUBBLE SORT
start = clock();
bubbleSortRecursive(arrForBubbleSort, n);
end = clock();
cpu_time_used = ((double) (end - start)) / CLOCKS_PER_SEC;
printf("Sorted array in %f seconds with BUBBLE sort: \n",cpu_time_used);
//   printArray(arrForBubbleSort, n);


//INSERTION SORT
start = clock();
insertionSortRecursive(arrForInsertionSort, n);
end = clock();
cpu_time_used = ((double) (end - start)) / CLOCKS_PER_SEC;

printf("Sorted array in %f seconds with INSERTION sort: \n",cpu_time_used);
//   printArray(arrForInsertionSort, n);

//SELECTION SORT
start = clock();
selectionSortRecursive(arrForSelectionSort, n, 0);
end = clock();
cpu_time_used = ((double) (end - start)) / CLOCKS_PER_SEC;

printf("Sorted array in %f seconds with SELECTION sort: \n",cpu_time_used);
//   printArray(arrForSelectionSort, n);

return 0;
}
```

- **Execution time in seconds :**

No.	Iterative			Recursive		
	Bubble	Insertion	Selection	Bubble	Insertion	Selection
500	0.000510	0.000165	0.000270	0.000742	0.000215	0.000945
1000	0.002620	0.000654	0.001059	0.003003	0.000793	0.003254
5000	0.068855	0.015397	0.022230	0.066351	0.014597	0.066256
10000	0.297178	0.061663	0.095448	0.294181	0.059248	0.260994
50000	7.761715	1.456488	2.327846	7.592220	1.411018	7.104381
100000	30.664242	5.497099	9.044861	31.578602	5.948873	31.051441
150000	69.463882	13.294161	20.671596	68.497410	-	-
160000	81.232848	14.830330	23.263657	-	-	-
170000	91.833924	16.277765	25.831034	-	-	-
200000	126.116330	23.078694	36.268727	-	-	-
500000	791.335	143.117	231.829	-	-	-

Output snapshots are attached on the next page

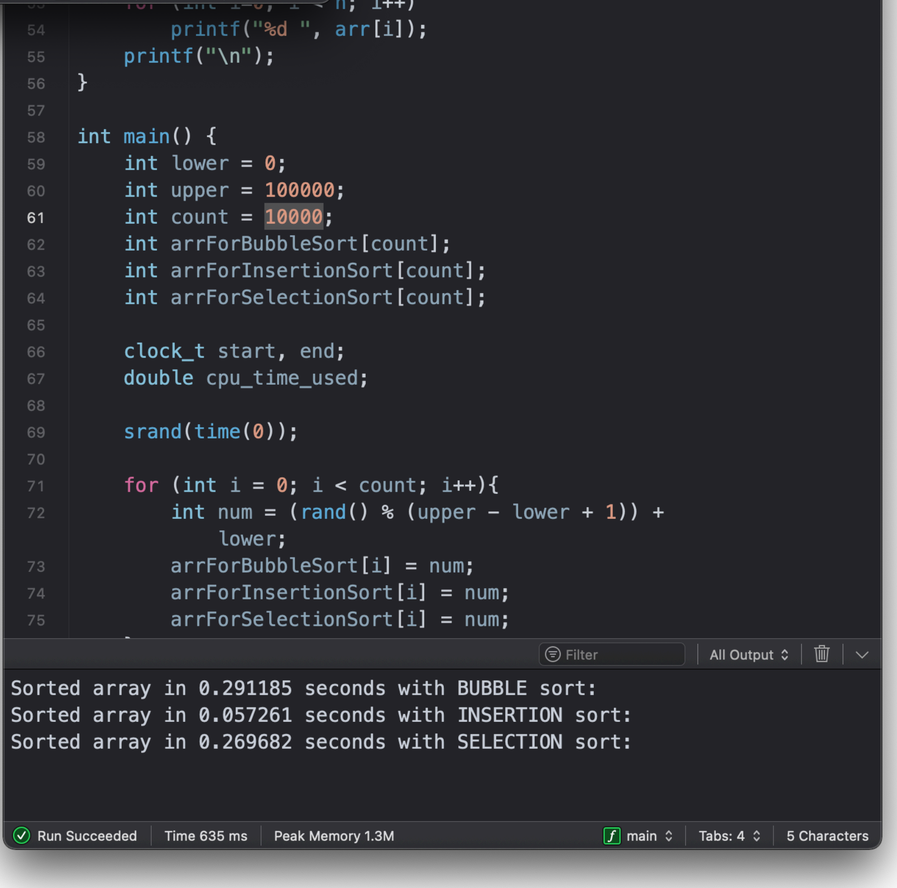


```c
            min_idx = j;
        swap(&arr[min_idx], &arr[i]);
    }
}

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

int main() {
    int lower = 0;
    int upper = 100000;
    int count = 160000;
    int arrForBubbleSort[count];
    int arrForInsertionSort[count];
    int arrForSelectionSort[count];

    clock_t start, end;
    double cpu_time_used;

    srand(time(0));

    for (int i = 0; i < count; i++){
        int num = (rand() % (upper - lower + 1)) + lower;
        arrForBubbleSort[i] = num;
```

```
Sorted array in 81.232848 seconds with BUBBLE sort:
Sorted array in 14.830330 seconds with INSERTION sort:
Sorted array in 23.263657 seconds with SELECTION sort:
```

Iterative Approach

Recursive Approach



```c
r[], int n)

n; i++)
        printf("%d ", arr[i]);
    printf("\n");
}

int main() {
    int lower = 0;
    int upper = 100000;
    int count = 10000;
    int arrForBubbleSort[count];
    int arrForInsertionSort[count];
    int arrForSelectionSort[count];

    clock_t start, end;
    double cpu_time_used;

    srand(time(0));

    for (int i = 0; i < count; i++){
        int num = (rand() % (upper - lower + 1)) +
            lower;
        arrForBubbleSort[i] = num;
        arrForInsertionSort[i] = num;
        arrForSelectionSort[i] = num;
```

```
Sorted array in 0.291185 seconds with BUBBLE sort:
Sorted array in 0.057261 seconds with INSERTION sort:
Sorted array in 0.269682 seconds with SELECTION sort:
```