**Assignment 2: Sorting**

Department of Computer Engineering

Dr Mélanie Bouroche

**Submitted By:**

Aniket Agarwal

Student ID: 17317437

**TASK 1:**

My approach to the problem:

* Generate arrays for all types of data
* Sort the list according the quicksort
* Calculating the **No. of Swaps** and **No. of comparisons** and printing it.
* **Time Complexity and space complexity for this algorithm is O(nlogn).**
* **No. of Swaps and No. of comparisons should be in the multiple of O(nlogn).**
* **All the results observed were within the limits**
* **Time Taken** for the program to run for 10000 values was **0.09 secs**.
* Algorithm used for Quicksort:

int Partition(int arr[],int i, int j, int pivot)

{

while(compare(i,j))

{

while(compare1(arr[i],pivot)) i++;

while(compare2(arr[j],pivot)) j--;

if(compare(i,j))

{

if(arr[i]!=arr[j]) swaping(&arr[i],&arr[j]) ;

i++;

j--;

}

}

return i;

}

void quickSort(int arr[], int low, int high)

{

if (low<high)

{

int median= arr[(low + high)/2];

int index= Partition(arr, low, high, median);

if(index>low) quickSort(arr, low, index-1);

if(index<high) quickSort(arr,index, high);

}

}

**TASK 2:**

My approach to the problem:

* General approach to the problem remains the same as was in the task 1.
* Used Selection sort for sorting.
* **Time Complexity and space complexity for this algorithm is O(n2).**
* **No. of Swaps and No. of comparisons should be in the multiple of O(n2).**
* **All the results observed were within the limits**
* **Time Taken** for the program to run for 10000 values was **1.24 secs**.
* No. of Swaps and comparisons observed using the selection sort were less compared to the quicksort but at the same time selection sort is much slower and time consuming. Also selection sort is unstable and not in-place sort, whereas quicksort is none of these and much faster sort.
* Just to verify this I ran a test run over 100,000 using both the sorts where selection sort almost took 112 sec but quick sort did it in less than 1 sec which I believe is pretty awesome!
* Algorithm used:

void selectionSort(int a[], int n)

{

int i, j, min;

for(i = 1; i < n+1; i++)

{

min = i;

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

if(compare(a[j],a[min]))

min = j;

swapping(&a[min],&a[i]);

}

}

**TASK 3:**

My approach to the problem:

* Retrieved data from the file ign
* Create a linked list
* Sort the linked list using the insertion sort
* Reason I chose Insertion sort was because insertion sort is one of the stable sorts and while using this sorting method all the results achieved by me were stable and correct to the best of my knowledge.
* Combined sorts to display the values asked in the sorts
* Algorithm used for sorting the linked list:

void insertion(struct ign\*\* sorted, struct ign\* cur)

{

struct ign\* newcur;

if (\*sorted == NULL || (\*sorted)->score <= cur->score)

{

cur->next = \*sorted;

\*sorted = cur;

}

else// Locate the node before the point of insertion

{

newcur = \*sorted;

while (newcur->next!=NULL && newcur->score >= cur->score)

newcur = newcur->next;

cur->next = newcur->next;

newcur->next = cur;

}

}

void sort(struct ign \*\*head)

{

struct ign \*sorted = NULL;

struct ign \*cur = \*head;

while (cur != NULL)

{

struct ign \*Next = cur->next; // Store next for next iteration

insertion(&sorted, cur); // insert cur in sorted linked list

cur = Next; // Update cur

}

\*head = sorted; // Update head to point to sorted linked list

}