JAYANTH SIVAKUMAR
B00615297
jsivaku1@binghamton.edu
BATCH: 12.00 P.M – 1.00 P.M

PROGRAMMING ASSIGNMENT 1

I have done this assignment completely on my own. I have not copied it, nor have I given my solution to anyone else. I understand that if I am involved in plagiarism or cheating I will have to sign an official form that I have cheated and that this form will be stored in my official university record. I also understand that I will receive a grade of **0** for the involved assignment for my first offense and that I will receive a grade of **"F" for the course** for any additional offense

PROBLEM 2:

Instruction count for the designed algorithm for problem 1:

INSERTION SORT:

```
//Insertion sort algorithm for sorting the randomly generated numbers
void insertion_sort(int n)
      {
            int i,j, temp;
            long int *number;
            if(n<=20){
                  number = generate_array_15(n);
            }
            else
                  number = generate_array(n);
            if(n<=20){
                  visualize(number, n);
            }
            for (i = 0 ; i <n ; i++) {
                  while ( i > 0 && number[i] < number[i-1]) {
                        temp          = number[i];
                        number[i]    = number[i-1];
                        number[i-1] = temp;
                        i--;
                  }
                  if(n<=20){
                        visualize(number, n);
                  }
            }
            printf("\nThe Sorted Array is:  \n");
            for (i = 0; i <n; i++) {
                  printf("%ld\t", number[i]);
            }
            printf("\n\n");
      }
```

Using Method 2 - Barometer operation

The instruction count for the insertion sort is $\sum_{i=0}^{n-1} \sum_{j=1}^{i} 1$

$$\sum_{i=1}^{n}\sum_{j=1}^{i} 1 = \sum_{i=1}^{n} i = 1 + 2 + 3 + \cdots .. + (n-1) + n = \frac{n(n-1)}{2}$$

Hence the above algorithm has instruction count $\frac{n(n-1)}{2}$. The insertion sort algorithm has the order $\Theta(n^2)$.

$$\textit{The Instruction count for insertion sort is } \Theta(n^2)$$

COUNTING SORT:

```
//Counting sort algorithm for randomly generated numbers between 0 to
99
void counting_sort(int n)
      {
            int i, j, curr = 0;
            long int *number;
            if(n<=20){
                  number = generate_array_15(n);
            }
            else
                  number = generate_array_countingsort(n);
            if(n<=20){
                        visualize(number, n);
                  }
            int max = maximum(number, n);
            int  *counting_array = calloc(max, sizeof(int));
            for(curr= 0;curr < n; curr++){
                  counting_array[number[curr]]++;
            }
            int num = 0;
            curr = 0;
            while(curr <= n){
                  while(counting_array[num] > 0){
                        number[curr] = num;
                        counting_array[num]--;
                        curr++;
                        if(curr > n){ break; }
                  }
                  if(n<=20){
                        visualize(number, n);
                  }
                  num++;
            }
            printf("\nThe Sorted array is: \n");
            for(curr = 0; curr < n; curr++){
                  printf("%ld\t ", number[curr]);
            }
            printf("\n\n"); }
```

Using Method 2 by Barometer operation

The instruction count for counting sort algorithm is $\sum_{0}^{n-1} \boldsymbol{count\_of\_numbers}$

$\sum_{1}^{n} \boldsymbol{counting\_array}[\boldsymbol{num}] = \boldsymbol{n}$

Hence the above algorithm has instruction count n. The counting sort algorithm has the order $\Theta(n)$.

$$\boldsymbol{The\ instruction\ count\ for\ counting\ sort\ algorithm\ is\ \theta(n)}$$

MERGE SORT:

```
void merge_sort(long int *number, int n, int inputdata)
      {
            long int mid;
            int i,j;
            long int *L, *R;
            if(n<2) return;
            mid = n/2;
            L=(long int*)malloc(mid*sizeof(long int));
            R=(long int*)malloc((n-mid)*sizeof(long int));
            for(i = 0;i<mid;i++) L[i] = number[i]; // creating left
subarray
            for(i = mid;i<n;i++) R[i-mid] = number[i]; // creating
right subarray
            merge_sort(L,mid, inputdata);  // sorting the left
subarray
            merge_sort(R,n-mid, inputdata);  // sorting the right
subarray
            Merge(number,L,mid,R,n-mid);  // Merging L and R into A as
sorted list.
            if(inputdata<=20){
                  visualize(number, inputdata);
            }
            free(L);
            free(R);
      }
```

The three recursion call is being made as follows:

```
MergeSort(L,mid);  // sorting the left subarray
MergeSort(R,n-mid);  // sorting the right subarray
```

```
Merge(number,L,mid,R,n-mid, n);  // Merging L and R into A as
sorted list.
```

Divide:

The divide step just computes the middle of the subarray, which takes constant time. Thus, $D(n) = \theta(1)$.

Conquer:

We recursively solve two sub problems, each of size $\frac{n}{2}$, which contributes $2T\left(\frac{n}{2}\right)$ to the running time.

Combine:

We have already noted that the MERGE procedure on an n-element subarray takes time , $\theta(n)$ and so $C(n) = \theta(n)$.

When we add the functions $D(n) and\ C(n)$ for the merge sort analysis, we are adding a function that is $\theta(n)$ and a function that is $\theta(1)$. This sum is a linear function of n, that is $\theta(n)$. Adding it to the $2T(\frac{n}{2})$ term from the "conquer" step gives the recurrence for the worst-case running time T(n) of merge sort:

The number of comparisons can be given by the master theorem.

$$\boldsymbol{T(n) = \theta(1) \quad if\ n = 1}$$

$$\boldsymbol{T(n) = 2T\left(\frac{n}{2}\right) + \theta(n) \quad if\ n > 1}$$

For the convenience of calculating the instruction sort, we assume the master theorem for merge sort as,

$$T(1) = 1 \quad if\ n = 1$$

$$T(n) = 2T\left(\frac{n}{2}\right) + n \quad if\ n > 1$$

$$Assume\ n = 2^k$$

$$= 2T\left(\frac{n}{2}\right) + n$$

$$= 2\left(2T\left(\frac{n}{4}\right) + \frac{n}{2}\right) + n$$

$$= \left(4T\left(\frac{n}{4}\right) + n + n\right)$$

$$= 4(2T\left(\frac{n}{8}\right) + \frac{n}{4}) + 2n$$

$$= 8T\left(\frac{n}{8}\right) + n + 2n$$

… … … … … ….

$$= \ 2^k T\left(\frac{n}{2^k}\right) + kn$$

$$Here\ n = 2^k$$

$$= nT\left(\frac{n}{n}\right) + kn$$

$$= \ nT(1) + kn$$

$$T(1) = 1$$

$$= \ n + kn$$

$$= n + \mathrm{nlog}\, n \qquad (since, kn = n \log n)$$

$$= \boldsymbol{n} + \boldsymbol{\theta}(\boldsymbol{n} \log \boldsymbol{n})$$

$$\boldsymbol{The\ instruction\ count\ for\ merge\ sort\ is\ \theta}(\boldsymbol{n} \log \boldsymbol{n})$$