EECE 7204-Assignment1

Sreejith Sreekumar

September 24, 2018

1 Coding

1.1 Insertion Sort

```
#include <iostream>
#include <time.h>


int *get_input(int limit){

   /**
    * Generating input - Worst case happens when the array is sorted in
    * reverse order. Sending numbers from 1000 to limit
    * decrementing by 1 for every element.
    */

  int* input = new int[limit];

  int j = 0;
  for(int i=1000; i>1000-limit; i--){
    input[j] = i;
    j+=1;
  }
  return input;
}


int *insertion_sort(int input[], int limit){

   /**
    * Insertion sort algorithm
    */

  int i, j, temp;
  for (i=0; i< limit; i++){
    for (j=i ;j>=0; j--){
      if(input[j] < input[j-1]){
```

```
        temp = input[j];
        input[j] = input[j-1];
        input[j-1] = temp;
      }
    }
  }
  return input;
}


int main(){

  int limit, i;
  std :: cout <<"Enter limit: ";
  std :: cin>> limit;

  int* input = get_input(limit);

  /*
   * Setting clocks before and after the sorting
   */
  clock_t start = clock();
  int* sorted_array = insertion_sort(input, limit);
  clock_t end = clock();

  double cpu_time_used = ((double) (end - start)) / CLOCKS_PER_SEC;

  printf("Insertion Sort took %f seconds to finish \n", cpu_time_used);

}
```

1.2 Merge Sort

```
#include <iostream>
#include <time.h>

int *get_input(int limit){

   /**
    * Generating input - Worst case - Sending numbers from 1000 to limit
    * decrementing by 1 for every element.
    */

  int* input = new int[limit];

  int j = 0;
  for(int i=1000; i>1000-limit; i--){
    input[j] = i;
```

```
    j+=1;
  }
  return input;
}


void copy_merged_array_into_original(int original[], int merged[], int length, int left_low) {

    for (int i=0; i< length; ++i)
        original[left_low++] = merged[i];

}


void merge(int input[], int left_low, int left_high, int right_low, int right_high) {

    int length = right_high-left_low+1;
    int merged_array[length];
    int left = left_low;
    int right = right_low;

    auto left_array_exhausted = [&left, &left_high]() { return left > left_high;};
    auto right_array_exhausted = [&right, &right_high]() { return right > right_high;};

    for (int i = 0; i < length; ++i) {
        if (left_array_exhausted())
            merged_array[i] = input[right++];
        else if (right_array_exhausted())
            merged_array[i] = input[left++];
        else if (input[left] <= input[right])
            merged_array[i] = input[left++];
        else
            merged_array[i] = input[right++];
    }
    copy_merged_array_into_original(input,  merged_array, length, left_low);

}


void merge_sort(int numbers[], int low, int high) {

    if (low >= high)
        return;

    else {

        /*
         * Recursive sorting and merging parts
```

```
         */

        int mid = (low + high)/2;
        merge_sort(numbers, low, mid);
        merge_sort(numbers, mid+1, high);
        merge(numbers, low, mid, mid+1, high);
    }

}


int main(){


    int limit;

    std :: cout << "Enter limit: ";
    std :: cin >> limit;

    int* input = get_input(limit);
    std :: cout << "Sorting using mergesort: ";

  /*
   * Setting clocks before and after the sorting
   */
    clock_t start = clock();
    merge_sort(input, 0, limit-1);
    clock_t end = clock();

    double cpu_time_used = ((double) (end - start)) / CLOCKS_PER_SEC;
    printf("Merge Sort took %f seconds to finish \n", cpu_time_used);


}
```

2 Arrangement of Elements during Sorting

2.1 Insertion Sort

```
Input: {10, 5, 7, 9, 8, 3}
Iteration 1: {5, 10, 7, 9, 8, 3}
Iteration 2: {5, 7, 10, 9, 8, 3}
Iteration 3: {5, 7, 9, 10, 8, 3}
Iteration 4: {5, 7, 9, 8, 10, 3}
             {5, 7, 8, 9, 10, 3}
Iteration 5: {5, 7, 8, 9, 3, 10}
             {5, 7, 8, 3, 9, 10}
             {5, 7, 3, 8, 9, 10}
```

```
{5, 3, 7, 8, 9, 10}
{3, 5, 7, 8, 9, 10}
```

2.2 Quicksort

Input: 10, 5, 7, 9, 8, 3 Iteration 1: 10

3 True or False

- n + 3 $\in \Omega$(n) - True, n+3 is lower bounded by Ω(n)
- n + 3 $\in \Omega(n^2)$ - False, n+3 is not lower bounded by $\Omega(n^2)$
- n + 3 $\in \Theta(n^2)$ - False
- $2^{n+1} \in \mathcal{O}$(n + 1) - False
- $2^{n+1} \in \Theta(2^n)$ - True

4 Master Method

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

  Here a = 8, b = 2 and f(n) = n
  Considering Case I of the Master Method, i,e $\mathcal{O}(n^{\log_b a - \varepsilon})$ and substituting for a and b, we have:

  f(n) = n = $\mathcal{O}(n^{\log_2 8-\varepsilon}) = \mathcal{O}(n^{3-\varepsilon})$

  For $\varepsilon = 1$, we have f(n) = $\mathcal{O}(n^2)$, which satisfies Case I

  $\therefore$ T(n) = $\Theta(n^{\log_b a}) = \Theta(n^3)$

- T(n) = 8T($\frac{n}{2}$) + n^2

  Here a = 8, b = 2 and f(n) = n^2
  Considering Case II of the Master Method, i,e $\Theta(n^{\log_b a})$ and substituting for a and b, we have:

  f(n) = n^2 = $\Theta(n^{\log_2 8}) = \Theta(n^3)$, which satisfies Case II

  $\therefore$ T(n) = $\Theta(n^{\log_b a}.log^{k+1}.n) = \Theta(n^3.log\ n)$

- T(n) = 8T($\frac{n}{2}$) + n^3

  Here a = 8, b = 2 and f(n) = n^3
  Considering Case III of the Master Method, i,e $\Theta(n^{\log_b a})$ and substituting for a and b, we have:

$$f(n) = n^3 = \Omega(n^{\log_2 8+\varepsilon}) = \Omega(n^{3+\varepsilon}) \tag{1}$$

and,

$$8.f(\frac{n}{2}) \leq (1-\varepsilon').n^3 \tag{2}$$

$$\implies 8.\frac{n^3}{8} \leq (1-\varepsilon').n^3 \tag{3}$$

$$\implies n^3 \leq (1-\varepsilon').n^3 \tag{4}$$

(4) is true for many $\varepsilon' s > 0$

From (1) and (4) Master Method Case III is satisfied.

$\therefore$ T(n) = Θ(f(n)) = $\Theta(n^3)$

- T(n) = 8T($\frac{n}{2}$) + n^4