Fundamentals of Computer Engineering

Homework - 1

Q1. Comparing Insertion & merge sort for different values of n

```cpp
#include <iostream>
#include <vector>
#include <ctime>

void insertionSort(std::vector<int>& arr) {
    int n = arr.size();
    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;
        }
        arr[j + 1] = key;
    }
}
```

Insertion sort
Worst case : $O(n^2)$
Reversely sorted array.

```cpp
#include <iostream>
#include <vector>
#include <ctime>

void merge(std::vector<int>& arr, int l, int m, int r) {
    int n1 = m - l + 1;
    int n2 = r - m;

    std::vector<int> L(n1), R(n2);

    for (int i = 0; i < n1; ++i)
        L[i] = arr[l + i];
    for (int i = 0; i < n2; ++i)
        R[i] = arr[m + 1 + i];

    int i = 0, j = 0, k = l;
    while (i < n1 && j < n2) {
        if (L[i] <= R[j])
            arr[k++] = L[i++];
        else
            arr[k++] = R[j++];
    }

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

void mergeSort(std::vector<int>& arr, int l, int r) {
    if (l >= r) return;
    int m = l + (r - l) / 2;
    mergeSort(arr, l, m);
    mergeSort(arr, m + 1, r);
    merge(arr, l, m, r);
}
```

Merge Sort
Worse Case :
$T(n) = n \log n$
(regardless of input)

[Consider the same reverse array for both

main

```cpp
#include <iostream>
#include <vector>
#include <ctime>
#include <cstdlib>

void insertionSort(std::vector<int>& arr);
void mergeSort(std::vector<int>& arr, int l, int r);
std::vector<int> generateWorstCaseInput(int n);
void measureAndReport(int n);

int main() {
    for (int n = 100; n <= 300; n += 20) {
        measureAndReport(n);
    }
    return 0;
}


std::vector<int> generateWorstCaseInput(int n) {
    std::vector<int> arr(n);
    for (int i = 0; i < n; ++i) {
        arr[i] = n - i;
    }
    return arr;
}

// Measure running time and report results
void measureAndReport(int n) {
    std::vector<int> arr1 = generateWorstCaseInput(n);
    std::vector<int> arr2 = arr1;

    clock_t start, end;
    double timeInsertion, timeMerge;

    // Measure Insertion Sort
    start = clock();
    insertionSort(arr1);
    end = clock();
    timeInsertion = double(end - start) / CLOCKS_PER_SEC;

    // Measure Merge Sort
    start = clock();
    mergeSort(arr2, 0, arr2.size() - 1);
    end = clock();
    timeMerge = double(end - start) / CLOCKS_PER_SEC;

    // Report results
    std::cout << "Input Size n = " << n << std::endl;
    std::cout << "Insertion Sort Time: " << timeInsertion << " seconds" << std::endl;
    std::cout << "Merge Sort Time: " << timeMerge << " seconds" << std::endl;
    std::cout << std::endl;
}

```

```
Input Size n = 100
Insertion Sort Time: 3.2e-05 seconds
Merge Sort Time: 3.8e-05 seconds

Input Size n = 120
Insertion Sort Time: 4.8e-05 seconds
Merge Sort Time: 5.8e-05 seconds

Input Size n = 140
Insertion Sort Time: 9e-05 seconds
Merge Sort Time: 8.4e-05 seconds

Input Size n = 160
Insertion Sort Time: 7.8e-05 seconds
Merge Sort Time: 5.7e-05 seconds

Input Size n = 180
Insertion Sort Time: 0.000101 seconds
Merge Sort Time: 6.7e-05 seconds

Input Size n = 200
Insertion Sort Time: 0.000125 seconds
Merge Sort Time: 7.5e-05 seconds

Input Size n = 220
Insertion Sort Time: 0.000147 seconds
Merge Sort Time: 8.2e-05 seconds

Input Size n = 240
Insertion Sort Time: 0.000209 seconds
Merge Sort Time: 0.000106 seconds

Input Size n = 260
Insertion Sort Time: 0.000251 seconds
Merge Sort Time: 0.000137 seconds

Input Size n = 280
Insertion Sort Time: 0.000235 seconds
Merge Sort Time: 0.000111 seconds

Input Size n = 300
Insertion Sort Time: 0.000309 seconds
Merge Sort Time: 0.000115 seconds
```

RESULT

We observe that insertion sort is faster than merge sort for smaller values of n

As n grows, the quadratic nature of insertion sort will make it slower.

Merge Sort starts to beat insertion sort typically around n = 160 to 300

eg. at n = 160 insertion sort took 0.000078 seconds and merge sort took 0.000057 seconds.
so at n = 160 merge sort started outperforming insertion sort.

Q2. Quick Sort Algorithm

```cpp
#include <iostream>
#include <vector>
#include <ctime>
#include <algorithm>


int partition(std::vector<int>& arr, int low, int high) {
    int pivot = arr[high]; // pivot
    int i = (low - 1); // Index of smaller element

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

void quickSort(std::vector<int>& arr, int low, int high) {
    if (low < high) {
        int pi = partition(arr, low, high);
        quickSort(arr, low, pi - 1);
        quickSort(arr, pi + 1, high);
    }
}

// Function to generate sorted worst-case input
std::vector<int> generateWorstCaseInput(int n) {
    std::vector<int> arr(n);
    for (int i = 0; i < n; ++i) {
        arr[i] = i + 1; // Sorted array
    }
    return arr;
}

// Function to shuffle the array for average case
void shuffleInput(std::vector<int>& arr) {
    std::random_shuffle(arr.begin(), arr.end());
}
```

```cpp

// Measure and report running times
void measureAndReport(int n) {
    std::vector<int> worstCaseArray = generateWorstCaseInput(n);
    std::vector<int> shuffledArray = worstCaseArray; // Copy to shuffle later

    clock_t start, end;
    double timeWorstCase, timeShuffled;

    // Measure Quicksort for worst-case input (sorted array)
    start = clock();
    quickSort(worstCaseArray, 0, worstCaseArray.size() - 1);
    end = clock();
    timeWorstCase = double(end - start) / CLOCKS_PER_SEC;

    // Shuffle the array for average case
    shuffleInput(shuffledArray);

    // Measure Quicksort for shuffled input
    start = clock();
    quickSort(shuffledArray, 0, shuffledArray.size() - 1);
    end = clock();
    timeShuffled = double(end - start) / CLOCKS_PER_SEC;

    // Report the results
    std::cout << "Input Size n = " << n << std::endl;
    std::cout << "Quicksort Worst-Case Time (sorted input): " << timeWorstCase << " seconds" << std::endl;
    std::cout << "Quicksort Average-Case Time (shuffled input): " << timeShuffled << " seconds" << std::endl;
    std::cout << std::endl;
}

int main() {
    // Testing Quicksort with sufficiently large n
    int n = 100000;
    measureAndReport(n);

    return 0;
}
```

```
Input Size n = 100000
Quicksort Worst-Case Time (sorted input): 62.9601 seconds
Quicksort Average-Case Time (shuffled input): 0.025673 seconds
```

Quicksort Worse Case is $O(n^2)$ which occurs when the pivot divides the array unevenly (sorted or reversely sorted array)
Shuffling the array gives the average time complexity of $O(n \log n)$

Q3. True or False.

i> $n+3 \in \Omega(n)$

$\Omega(g(n)) = \{f(n)$: there exists constants $c>0$, $n>0$, such that $0 \leq c(g(n)) \leq f(n)$ for all $n \geq n_0\}$

$g(n) = n$

$f(n) = n+3$

$0 \leq cn \leq n+3$

$0 \leq cn \rightarrow$ Satisfied

$cn \leq n+3$

if $c = 1 \qquad n_0 = 1$

$1 \leq 4$.

$0 \leq n \leq n+3 \rightarrow$ Satisfied.

TRUE

ii> $n+3 \in O(n^2)$

$O(g(n)) = \{f(n)$: there exists constants $c>0$, $n>0$, such that $0 \leq f(n) \leq cg(n)$ for all $n \geq n_0\}$

$f(n) = n+3 \qquad g(n) = n^2$

$0 \leq n+3 \leq cn^2$

$0 \leq n+3 \rightarrow$ Satisfied.

$n+3 \leq cn^2$

$3 \leq cn^2 - n$

$3 \leq n(n-1)$

if $c = 1 \quad n_0 = 3$

$6 \leq 9$

$0 \leq n+3 \leq n^2 \rightarrow$ Satisfied

TRUE

iii) $n+3 \in \Theta(n^2)$

$\Theta(g(n)) = \Theta(g(n)) \cap \Omega(g(n))$

$\Theta(g(n)) = \{f(n)$: there exists constants c_1, c_2, n_0 such that $0 \leq c_1 g(n) \leq f(n) \leq c_2 g(n)$ for all $n \geq n_0\}$

$g(n) = n^2 \qquad f(n) = n+3$

$0 \leq c_1 n^2 \leq n+3 \leq c_2 n^2$

if $c_1 = 1, c_2 = 5, n_0 = 1$

then $0 \leq n^2 \leq n+3 \leq 5n^2$

$0 \leq 1 \leq 4 \leq 5 \rightarrow$ Satisfied

TRUE

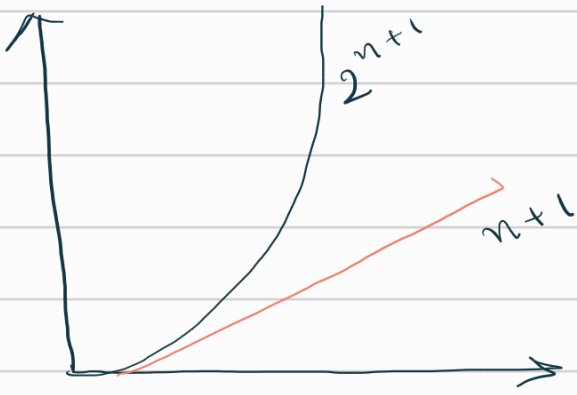
iv) $2^{n+1} \in O(n+1)$

$g(n) = n+1$

$f(n) = 2^{n+1}$

$$0 \leq f(n) \leq c\,g(n)$$

$$0 \leq 2^{n+1} \leq c(n+1)$$



from the graph we can see that $g(n)$ can't be the upper bound of $f(n)$

$c = 1 \quad n_0 = 1$

$0 \leq 2^{2^0} \leq 2$ Not Satisfied

FALSE

$v > 2^{n+1} \in O(2^n)$

$g(n) = 2^n$

$f(n) = 2^{n+1}$

$0 \leq 2^{n+1} \leq c2^n$

$2 \cdot 2^n \leq c2^n$

$c = 2 \quad n = 1$ satisfies function

↳ same functions

TRUE

Q4. Master's Theorem

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

$\log_b a = 3$, $k = 1$, $3 > 1$, $\Theta(n^{\log_b a})$

Master method applies for recurrence of the form $T(n) = aT\left(\frac{n}{b}\right) + f(n)$ where $a \geq 1$, $b > 1$ and $f(n)$ is asymptotically positive, $f(n) > 0$ for $n \geq n_0$

(compares $f(n)$ with $n^{\log_b a}$)

$f(n) = n \quad a = 8 \quad b = 2$

Compare $f(n)$ with $n^{\log_b a} = n^{\log_2 8} = n^3$

Case 1:

$f(n) = O(n^{\log_b a - \varepsilon}) \rightarrow T(n) = \Theta(n^{\log_b a})$

$n = O(n^{3-2}) \qquad 0 < \varepsilon \leq 2$

The above function is a form of Case 1:

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

$T(n) = \Theta(n^3)$

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

$\log_b a = 3$, $k = 2$, $3 > 2$, $\Theta(n^3)$

$a = 8, \quad b = 2 \qquad n^{\log_b a} = n^3$

$f(n) = n^2 \rightarrow O(n^{3-\varepsilon})$

$0 < \varepsilon \leq 1$

Case 1: $T(n) = \Theta(n^3)$

iii) $T(n) = 8T\left(\frac{n}{2}\right) + n^3$

$\log_b a = 3$
$k = 3$
$3 = 3$
$\Theta(n^3 \log n)$

$a = 8 \quad b = 2 \quad f(n) = n^3$

$n^{\log_b a} = n^3$

Case 2: $f(n) = \Theta(n^{\log_b a}) \rightarrow \Theta(n^{\log_b a} \log n)$

$$T(n) = \Theta(n^3 \log n)$$

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

$\log_b a = 3$
$k = 4$
$4 > 3$
$\Theta(n^4)$

$a = 8 \quad b = 2 \quad f(n) = n^4$

$n^{\log_b a} = n^3$

Case 3: $f(n) = \Omega(n^{\log_b a + \varepsilon})$ for $\varepsilon > 0$

and $af\left(\frac{n}{b}\right) \leq (1-\varepsilon') \cdot f(n)$ for $\varepsilon' > 0$

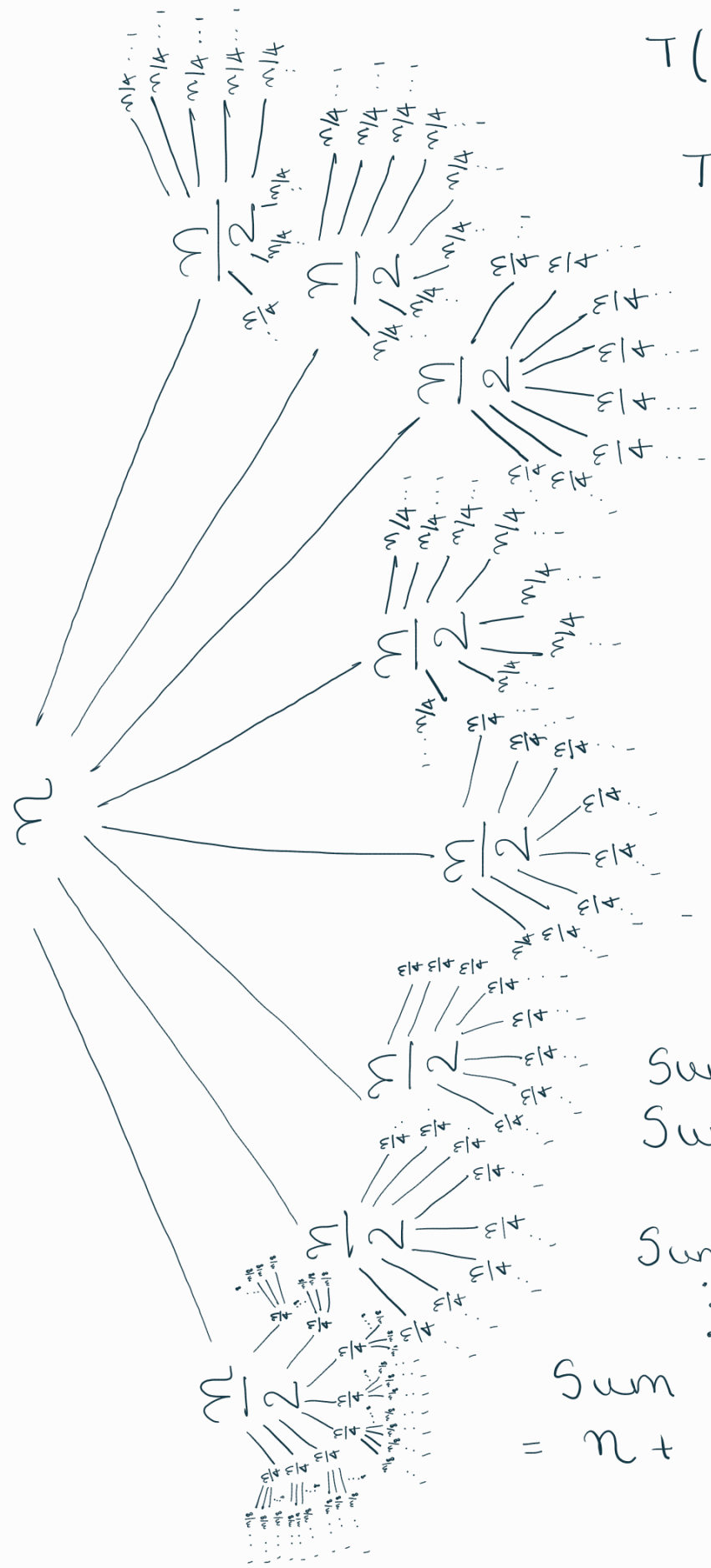
$$T(n) = \Theta(f(n))$$

$f(n) = n^4 = \Omega(n^{3+\varepsilon}) \qquad 0 < \varepsilon \leq 1$

Case 3: $T(n) = \Theta(n^4)$

Q5. Recursion tree for $T(n) = 8T\left(\frac{n}{2}\right) + n$ and prove by substitution

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



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

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

$$T(n/4) = 8T(n/16) + \frac{n}{4}$$

$$\vdots$$

$$T\left(\frac{n}{2^k}\right) = T(1)$$

$$n = 2^k$$

$$\text{height} = \log_2 n$$

Sum of leaf nodes

$$= n^{\log_b a} = n^{\log_2 8} = n^3$$

Sum at level 1 : n

Sum at level 2 : $8 \times \frac{n}{2} = 4n$

Sum at level 3 : $8 \times \frac{n}{4} \times 8 = 16n$

$$\vdots$$

Sum of internal nodes

$$= n + 4n + 16n + \ldots + (4)^{2k-1} n$$

Total Cost = Cost of leaf nodes + Cost of internal nodes

$$T(n) = n^3 + n$$

↘ negligible for smaller values of n

$$T(n) = \underline{\Theta(n^3)}$$

Using Substitution

Guess $T(n) = n^3$

$$T(n) = c8\left(\frac{n}{2}\right)^3 + n$$

$$cn^3 + n$$

n is negligible for small values and can be ignored.

$$T(n) = \underline{\Theta(n^3)}$$

PROVED