COMP 6651 ADT Winter 2023
Assignment 1
Submitted by: Rajat Sharma (40196467)

PROBLEMS.

1. Indicate, for each pair of expressions (A, B) in Table 1, whether A is O, o, Ω, ω, or Θ of B. Assume that k ≥ 1, ϵ > 0, and c > 1 are constants. Your answer should be in the form of the table with "yes" or "no" written in each box.

A	B	O	o	Ω	ω	Θ
$\log^k n$	n^ϵ	yes	yes	no	no	no
n^k	c^n	yes	yes	no	no	no
$\sqrt{n}$	n sin n	no	no	no	no	no
2^n	$2^{n/2}$	no	no	yes	yes	no
$n^{\log n}$	$c^{\log n}$	no	no	yes	yes	no
log(n!)	$\log(n^n)$	yes	no	yes	no	yes

2. For f, g: N → R we say that f is asymptotically equal to g (notation f(n) ~ g(n)) if $\lim_{n\to\infty} \frac{f(n)}{g(n)} = 1$. **Use Stirling's formula**

$$n! \sim \left(\frac{n}{e}\right)^n \sqrt{2\pi n}$$

to determine constants c and d such that $\frac{\binom{2n}{n}}{4^n} \sim cn^d$ **.**

Ans)

Expanding the choose (2n n) by putting the value of 2n in Stirling's formula we get

$$\frac{\left(\frac{2n}{e}\right)^{2n}\sqrt{2\pi 2n}}{\left(\frac{n}{e}\right)^n\sqrt{2\pi n} * \left(\frac{n}{e}\right)^n\sqrt{2\pi n} * 4^n}$$

On simplifying this we get

$$\frac{1}{\sqrt{\pi n}} = \frac{1}{\sqrt{\pi}} * n^{-\frac{1}{2}}$$

This gives the value of c to be $1/\sqrt{\pi}$ and d = -½.

3. Show that f(n) ln f(n) = Θ(n) implies f(n) = Θ(n/ ln n).

proved using the symmetry property of the Θ notation,

$$f(n) \ln f(n) = \Theta(n) \Rightarrow n = \Theta(f(n) \ln f(n)) \quad \ldots(1)$$

Taking logarithm (base 2) of (1),

$$\ln n = \Theta(\ln(f(n) \ln f(n))) = \Theta(\ln f(n) + \ln \ln f(n)) = \Theta(\ln f(n)) \quad \ldots.(2)$$

Dividing (1) by (2),

$$\frac{n}{\ln n} = \frac{\Theta(f(n) \ln f(n))}{\Theta(\ln f(n))} = \Theta\left(\frac{f(n) \ln f(n)}{\ln f(n)}\right) = \Theta(f(n))$$

Using symmetry property again,

$$f(n) = \Theta\left(\frac{n}{\ln n}\right)$$

4. Consider the searching problem:
Input: A sequence of n numbers A = {a1, . . . , an} and a value v.
Output: An index i such that v = A[i], or the special value NIL if v does not appear in A.

1. **Write pseudocode for linear search: which scans through the sequence, looking for v.**
2. **Use a loop invariant to prove that your algorithm is correct. Make sure that your loop invariant fulfills the three necessary properties.**
3. **What is the worst-case running time of linear search?**

Now, suppose the sequence A is sorted, then we can check the midpoint of the sequence against v and eliminate half of the sequence from further consideration. The binary search algorithm repeats this procedure, halving the size of the remaining portion of the sequence each time.

4. **Write pseudocode for binary search.**
5. **Argue that the worst-case running time of binary search is Θ(log n).**

Ans)

1)

Linear Search Pseudocode:

```
LinearSearch(A, v)
  for i = 1 to n
    if A[i] == v
      return i
  return NIL
```

2)Loop Invariant: At the start of each iteration of the loop, the variable i takes values from 1 to n, and the value A[1..i-1] is not equal to v.

Proof of Loop Invariant:
Initialization: At the start of the first iteration, i = 1, the loop invariant is true because A[1..0] is an empty sequence, and it is not equal to v.
Maintenance: At the start of each iteration if A[i] == v, the function returns i and terminates. If A[i] ≠ v, the loop invariant remains true because A[1..i-1] does not contain v, and the value of i increases.
Termination: When i reaches n + 1, the loop terminates, and the function returns NIL, which means v is not in A.

3)Worst-Case Running Time of Linear Search: The worst-case running time of linear search is Θ(n). In the worst case, the value v is not in the sequence, and the algorithm needs to scan through all n elements, taking n iterations.

4)

```
BinarySearch(A, v)
  left = 1
  right = n
  while left <= right
    mid = (left + right) / 2
    if A[mid] == v
      return mid
    else if A[mid] < v
      left = mid + 1
    else
      right = mid - 1
  return NIL
```

5)Worst-Case Running Time of Binary Search: The worst-case running time of binary search is Θ(log n). In each iteration of the loop, the size of the remaining portion of the sequence is reduced by half, making the number of iterations logarithmic in the size of the sequence. The time to perform each iteration is constant, so the total running time is Θ(log n).

5. Use the divide-and-conquer integer multiplication algorithm to multiply two integers X = 83645283 and Y = 75461934. You are supposed to write down the execution of the main steps.

To solve for x.y using efficient divide and conquer, you can use the Karatsuba algorithm, which is a fast multiplication algorithm based on divide-and-conquer. Here's how you can solve for x.y using the Karatsuba algorithm:

Divide each number into two equal parts:
X = 8364 + 5283, Y = 7546 + 1934.

Recursively compute the following three products:

p1 = 8364 * 7546 (recursively)

a. first halves (83 and 75)(**recursively**) = 6225
b. second halves (64 and 46)(**recursively**)=2944
c. first and second halves of each number (83 + 46) * (75 + 64)(**recursively**) = 129*139=17,931
d. Final Result = (a * 10^(4)) + b + ((c - a - b) * 10^(2)) = 63114774

p2 = 5283 * 1934 p3 = (8364 + 5283) * (7546 + 1934)
= (8364 * 7546) + (8364 * 1934) + (5283 * 7546) + (5283 * 1934)
= p1 + (8364 * 1934 + 5283 * 7546)

Calculate the final result Z = A * 10^8 + (C - A - B) * 10^4 + B.

Final answer: Z = 83645283 * 75461934 = **6312034825157322**

Note that this algorithm reduces the number of multiplications required to compute x.y by a factor of approximately 2 compared to the traditional long multiplication method, making it more efficient.

6. Suppose you have k sorted arrays, each with n elements, i.e., A1[1..n], A2[1..n], . . ., Ak[1..n]. You wish to merge them into a single sorted array of kn elements.
(1) Here is one strategy: use the Merge procedure from the lecture to merge the first two arrays, then merge in the third, then merge in the fourth, and so on. What is the time complexity of this algorithm in terms of k and n?
(2) Design a more efficient solution to this problem using divide and conquer. Provide a pseudocode, write down the recurrence, and solve it.

Ans)

(1) Time Complexity of the First Strategy: The time complexity of this algorithm is Θ(nk^2). In each iteration, the Merge procedure takes Θ(n) time, and there are k - 1 iteration, so the total time complexity is Θ(nk^2).

2)

```
MergeKArrays(A, l, r)
  if l == r
    return A[l]
  mid = (l + r) / 2
  L = MergeKArrays(A, l, mid)
  R = MergeKArrays(A, mid + 1, r)
  return Merge(L, R)
```

Recurrence: T(n) = 2T(n/2) + O(nk)

Time Complexity: The time complexity of this algorithm is Θ(nk log k). By the Master Theorem, the solution to the recurrence is T(n) = Θ(nk log k). The algorithm divides the problem into two subproblems of size n/2 in each iteration, and each subproblem takes O(k) time to merge the arrays. The total time complexity is Θ(nk log k).