Assignment Unit 5

Created by William Jiang, Tim Nadolsky

May 2024

1. Prove the Big-O complexity for Human Sort as described in the tutorial.

2. Indicate, for each pair of expressions (A, B) in the table below, whether A is O, Ω, or Θ, of B. Assume all logarithms have base 2.

a. $A = \log(n!), B = \log(n^n)$

b. $A = 400n^{-3} + \sqrt{n} + \pi, B = 1 + \sin\frac{n}{\pi}$

c. $A = \log(n)^{\log(n)}, B = 4^{\log(n)}$

d. $A = 2^n, B = n!$

e. $A = n^{-5099}, B = \frac{1}{5099}\log n$

3. Suppose the function $G : N \to N$ is defined recursively by setting $G(0) = 0$ and $G(n) = 1 + G(\lfloor n/2 \rfloor)$ for every positive integer n. Prove that for all $n \in \mathbb{N}$, we have $G(n) = 1 + \lfloor \log_2 n \rfloor$.

Hint: If you are struggling with the inductive step, try splitting the problem into even and odd cases.

4. Write three programs that calculate the nth prime number: one iterative (runs without calling itself and without tables), one using recursion, and one using dynamic programming.

Write a proof that your three algorithms are correct using the guidelines in the documents. Also give a tight Big-O bound for your algorithm and quickly discuss why it's correct.

You can use the Sieve of Eratosthenes as your prime-finding algorithm:

https://en.wikipedia.org/wiki/Sieve_of_Eratosthenes

For your proof, you may need the fact that every finite number has a prime factorization - that is, it can be factored into a product of a finite number of primes.

5. Consider a weighted graph G with V vertices and E edges, and suppose all edge weights are nonnegative. Prove using induction that the following algorithm can find the shortest path from a vertex "source" to any other vertex of the graph.

```
function Dijkstra(Graph, source):
    create vertex priority queue Q
    dist[source] ← 0
    add source to Q
    for each vertex v in Graph.Vertices:
        dist[v] ← INFINITY
        prev[v] ← UNDEFINED
        add v to Q
    dist[source] ← 0

    while Q is not empty:
        u ← vertex in Q with minimum dist[u]
        remove u from Q

        for each neighbor v of u still in Q:
            alt ← dist[u] + Graph.Edges(u, v)
            if alt < dist[v]:
                dist[v] ← alt
                prev[v] ← u

    return dist, prev
```

Also use Big-O notation to find the time and space complexity of the algorithm in terms of V and E. A short explanation will suffice.

6. Find a recurrence relation for the auxilliary space $S(n)$ used by mergesort on an array n. Use the recurrence to prove that the space complexity of merge sort is $\theta(n)$

Hint: After looking at the code, you might be tempted to think $S(n)$ satisfies the same recurrence as $T(n)$ from the tutorial. However, there is a slight difference that changes the space complexity.