Vikram Thirumaran

Prof. Thanh H. Nguyen

CS 315

March 2nd, 2025

HW6

Q1: Runtime Analysis

We need to determine the runtime of the given algorithms.

Q1.1: Toy Algorithm

Algorithm:

python
CopyEdit
```
procedure ToyAlg(n)
if n > 3 then
   ToyAlg(⌈n/3⌉)
   ToyAlg(⌈n/3⌉)
end if
```

Step 1: Formulating the Recurrence

The algorithm makes **two recursive calls**, each on an input of size **⌈n/3⌉**. Ignoring the ceiling function for now, we get the recurrence:

$T(n)=2T(n/3)+O(1)$

Using the **Master Theorem** (Case 3):

- $a=2$ (number of subproblems)
- $b=3$(division factor)
- $f(n)=O(1)$(constant work)

Compute $\log_b a = \log_3 2 \approx 0.63$

Since $f(n)=O(n^c)$ where $c=0$ and $c < \log_b a$, the recurrence follows Case 3:

$T(n) = O(n\^(\log_3 2)) \approx O(n^{0.63})$

Final Answer:

$$T(n)=O(n\^(\log_3 2))$$

Q1.2: Modification of Mergesort

Algorithm:

python
CopyEdit
procedure ModifiedSort(X)
n ← length of X
if n = 0 or n = 1 then
  return X
end if
left ← ModifiedSort(X[1 . . . ⌊n/2⌋])
right ← ModifiedSort(X[⌊n/2⌋+ 1 . . . n])
Y ← new list whose first half is left and second half is right
return Mergesort(Y)

Step 1: Formulating the Recurrence

- The algorithm **recursively sorts** the two halves $O(n\log n)$
- Then, it **concatenates** ($O(n)$).
- Finally, it **calls Mergesort again** on the entire array $O(n\log n)$
- $T(n)=2T(n/2)+O(n\log n)$

Using the Master Theorem:

- $a=2$, $b=2$, $f(n)=O(n\log n)$.
- Compare $f(n)$ with $O(n\^(\log_2 2)) = O(n)$.
- Since $O(n\log n)$ grows **slightly faster** than $O(n)$, by the Master Theorem:

$$T(n)=O(n\log^2 n))$$

Final Answer:

$$O(n\log^2 n)$$

Q2: Extension of Integer Multiplication

We need an **$O(n^{\wedge}(\log_2 3))$** algorithm to compute **10^n in binary**.

Step 1: Understanding the Binary Representation of 10^n

- In decimal: $10^n = 1$ followed by n zeros.
- In binary: 10^n is the power of **2 and 5**:

$$10^n = 2^n \times 5^n$$

- Computing 2^n is trivial in binary (just shift left).
- Computing 5^n in binary is non-trivial.

Step 2: Fast Exponentiation for 5n

- Use **divide and conquer multiplication (Karatsuba Algorithm)**.
- Compute 5^n efficiently using **Exponentiation by Squaring**:
  - If n is even: $5^n=(5^{n/2})^2$
  - If n is odd: $5^n=5 \times 5^{n-1}$

$$T(n)=3T(n/2)+O(n)$$

By the **Master Theorem** (Case 3):

$$T(n) = O(n^{\wedge}(\log_2 3))$$

Final Answer:

$$O(n^{\wedge}(\log_2 3))$$

Q3: Counting Individual Inversions

We need an **$O(n\log n)$** approach to find **C[i]**, the count of elements before index i

that are $\geq A[i]$.

Approach:

1. **Use a modified Merge Sort**:
   - Instead of sorting normally, count how many times a number from the right half is placed before numbers from the left half.
2. **Implementation Idea:**

- When merging two halves, every time an element from the right side is merged before a left-side element, it means that **all remaining elements in the left half** are greater than or equal.

$$T(n)=O(n\log n)$$

Final Answer:

$$O(n\log n)$$

Q4: Greedy Knapsack

We need an **O(n)** greedy approach where $w_i \geq \sum_{j=1}^{i-1} w_j$.

Key Insight:

- This structure implies that **each item is at least as large as the sum of all previous items**.
- Thus, **at most one item can be chosen at each step**.

Algorithm:

1. **Start from the heaviest item** and add it to the knapsack **until adding another item would exceed W**.
2. **Because of the weight constraint, at most O(n) operations are needed**.

Correctness Proof:

- The problem structure ensures that **choosing the largest valid item is always optimal**.

Final Answer:

$$O(n)$$