Midterm Exam: CSE4081 Analysis of Algorithms

Grant Butler, Computer Science, B.S.

1. Fundamental Ideas and Definitions

> Let $f = N \to R$ and $g = N \to R$ be *time complexity functions*.

1. What does it mean to say "$f(n)$ is big-*O* of $g(n)$?" Give a clear mathematical answer (You may translate to English but be precise and use correct descriptive words).

$f(N) = O(g(N))$ means that there are positive constants c and n_0 such that
$0 \leq f(N) \leq cg(N)$ for all $n \geq n_0$.

2. Prove that $f(n) = n^2 + n + 2$ is $O(n^4)$. Provide *witnesses* that establish your proof.

> $n^2 + n + 2 \leq cn^4$
> $n^2 + n + 2 \leq n^{4*}$
>
> test $n = 1$
> $(1)^2 + 1 + 2 \leq 1^4$
> $4 \leq 1$ FALSE
>
> test $n = 2$
> $2^2 + 2 + 2 \leq 2^4$
> $8 \leq 16$ TRUE

$\therefore f(N)$ is $O(N^4) | c = 1$ and $n_0 = 2$

*: assume $c = 1 \because$ we are trying to prove $O(N^4)$

2. Loops

2.1 Outcome

3. What is the value of sum after executing the code below and what is the time complexity of the code?

```
int snip(int m) {
    int sum = 0;
    for (int i = 0; i < m; i++) {
        sum = sum + pow(2, i);
    }
}
```

The loop is run N - 1 times.
∴ the time complexity of the function is *O*(N).

$$\sum_{i=1}^{m} = 2(2^m - 1) = 2^{m+1} - 2$$

∴ The value of the sum will always be $2^{m+1} - 2$.

Sample outputs of code:

```
m = 8

i        │       prev_sum        +       2^i      =       sum
─────────┼──────────────────────────────────────────────────────
1        │       0               +       2^1      =       2
2        │       2               +       2^2      =       6
3        │       6               +       2^3      =       14
4        │       14              +       2^4      =       30
5        │       30              +       2^5      =       62
6        │       62              +       2^6      =       126
7        │       126             +       2^7      =       254
─────────┴──────────────────────────────────────────────────────
                                                  sum:    254

m = 12

i        │       prev_sum        +       2^i      =       sum
─────────┼──────────────────────────────────────────────────────
1        │       0               +       2^1      =       2
2        │       2               +       2^2      =       6
3        │       6               +       2^3      =       14
4        │       14              +       2^4      =       30
5        │       30              +       2^5      =       62
6        │       62              +       2^6      =       126
7        │       126             +       2^7      =       254
8        │       254             +       2^8      =       510
9        │       510             +       2^9      =       1022
10       │       1022            +       2^10     =       2046
11       │       2046            +       2^11     =       4094
─────────┴──────────────────────────────────────────────────────
                                                  sum:    4094
```

4. What will be the value of sum after executing the code snippet below? Use summation notation to compute the time complexity of the code snippet below?

```
int sum = 0;
for (int i = 0; i < n; i++) {
    for (int j = n; j > 0; j--) {
        sum = sum + i * j;
    }
}
```

$\sum_{i=0}^{n} \sum_{j=1}^{n-1} (ij) = \frac{(n-1)(n^2)(n+1)}{4} = \frac{n^4-n^2}{4}$

$\therefore$ the time complexity of this function is $O(N^2)$.

Sample outputs of code:

```
n = 2

step     |i       |j       |sum     +       (i * j) =       sum
_________|________|________|_____________________________________
1        |0       |2       |0       +       (0 * 2) =       0
2        |0       |1       |0       +       (0 * 1) =       0
3        |1       |2       |0       +       (1 * 2) =       2
4        |1       |1       |2       +       (1 * 1) =       3
_________|________|________|_____________________________________

                                                sum:    3

n = 3

step     |i       |j       |sum     +       (i * j) =       sum
_________|________|________|_____________________________________
1        |0       |3       |0       +       (0 * 3) =       0
2        |0       |2       |0       +       (0 * 2) =       0
3        |0       |1       |0       +       (0 * 1) =       0
4        |1       |3       |0       +       (1 * 3) =       3
5        |1       |2       |3       +       (1 * 2) =       5
6        |1       |1       |5       +       (1 * 1) =       6
7        |2       |3       |6       +       (2 * 3) =       12
8        |2       |2       |12      +       (2 * 2) =       16
9        |2       |1       |16      +       (2 * 1) =       18
_________|________|________|_____________________________________

                                                sum:    18
```

3. Recurrences

3.1 Outcome

5. What initial conditions (or assumptions) must be True for *binary search* to execute correctly on an array of length *n*.

Initial conditions for *binary search* to to execute properly are:
Array must be sorted in order to choose the half of the array to go to next.

Also, what is the recurrence relation for binary search, and what is the solution of this recurrence?

$$
\begin{aligned}
T(n) &= T\left(\frac{n}{2}\right) + 1 \\
T\left(\frac{n}{2}\right) &= T\left(\frac{n}{4}\right) + 1 \\
T\left(\frac{n}{4}\right) &= T\left(\frac{n}{8}\right) + 1 \\
T\left(\frac{n}{2^{k-1}}\right) &= T\left(\frac{n}{2^k}\right) + 1(k) \\
\implies T(n) &= T\left(\frac{n}{2^k}\right) + 1(k) \\
\frac{n}{2^k} &= 1 \\
n &= 2^k \\
\log_2 n &= k \\
T(n) &= T(1) + \log_2 n \\
T(n) &= \log_2 n + 1 \\
&\implies O(\log n)
\end{aligned}
$$

$\therefore$ The time complexity of binary search is $O(\log n)$

4. Algorithms

4.1 Sorting Algorithms

6. (10 points) Quicksort is a classic sorting algorithm. Describe it's best- and worst-case running time for sorting an array of size n by giving it's best case and worst case recurrences together with their solutions.

Best Case: The pivot of the array is the median of the array, and $n-1$ comparisons happen.

$$
\begin{aligned}
T(n) &= 2T\left(\frac{n}{2}\right) + n - 1 \\
&= 2\left(2T\left(\frac{n}{4}\right) + \frac{n}{2} - 1\right) + n - 1 \\
&= 4\left(2T\left(\frac{n}{8}\right) + \frac{n}{4} - 1\right) + 2n - 3
\end{aligned}
$$

$$
\begin{aligned}
\implies T(n) &= 2^k T\left(\frac{n}{2^k}\right) + kn - (2^k - 1) - (c) \\
k &= \log_2 n \\
\implies T(n) &= n \log_2 n - n + 1
\end{aligned}
$$

$\therefore$ Time complexity is $O(n \log n)$

Worst Case: Pivot is at the end of the list. One empty list and list of length $n-1$ are made, and $n-1$ comparisons are made.

$$
\begin{aligned}
T(n) &= T(n-1) + n - 1 \\
&= T(n-2) + n - 1 + n - 2 \\
&= T(n-3) + n - 1 + n - 2 + n - 1 \\
&= T(n-3) + 3n - 6
\end{aligned}
$$

$$
\begin{aligned}
\implies T(n) &= T(n-k) - k \\
n &= k + 1 \rightarrow k = n - 1
\end{aligned}
$$

$$
\begin{aligned}
T(n) &= T(1) + \sum_{i=0}^{m-1} (n - i) \\
&= 0 + \frac{n(n-1)}{2}
\end{aligned}
$$

$\therefore$ Time complexity is $O(n^2)$

4.2 Searching Algorithms

7. How long will brute-force pattern matching take?

Worst Case: $O(mn)$

$$\begin{aligned} & mn = 5\times 10^6 * 3.2\times 10^9 = 1.6\times 10^{16} \\ \implies & \frac{1.6\times 10^{16} seconds}{31\times 10^6 seconds} = 5\times 10^8 years \end{aligned}$$

8. How long will Knuth-Morris-Pratt pattern matching take?

Worst Case: $O(m) + O(n)$

$$\begin{aligned} m + n = 3.2\times 10^9 + 5\times 10^6 = 3205\times 10^6 seconds \\ \implies \frac{3205\times 10^6}{32\times 10^6} \to 100.15625 years \end{aligned}$$

9. How long will the Boyer-Moore pattern matching algorithm take?

Worst Case: $O(m) + O(nm)$

$$\begin{aligned} m + nm = 3.2\times 10^9 + (5\times 10^6 * 3.2\times 10^9) \\ = (1.6\times 10^{16} + 3.2\times 10^9 seconds) * \frac{1 year}{32\times 10^6} = 500000100 years \end{aligned}$$

4.3 Algorithms in General

4.4 Outcome

10. List three or more paradigms that can be used to classify algorithms.

- Brute force: try every possible solution
- Divide and conquer: divide the problem into smaller ones with solutions that can be combined to solve the larger one.
- Dynamic programming: express a big problem in smaller problems that have already been solved and memorized.
- Greedy: globally optimal solutions from locally optimal choices.

11. Some problems only require a “yes” or “no” answer. Other problems require a list of feasible answers, with one (or more) that is best by some measure.
What are the common names of these problem types?

- Yes or No questions –> Binary questions or True/False questions.
- list of feasible answers –> Multiple choice questions