CSE101 Quiz3 Review

This review doc summarizes Divide and Conquer methods in lectures. It follows the section order to textbook to organize everything. Created by M. and Yilin, feel free to collaborate.

Exam Format

- **Short answer, True/False:** Master theorem, Expected runtime, Recursive runtime, quick select, KS multiply.
- **Design D/C**: design, justification, runtime.

CHAPTER 2
Divide and Conquer

- Strategy
  - Break problem into *subproblems* that are themselves smaller instances of the same type of problem
  - Recursively solving these subproblems
  - Appropriately combining their answers

2. 1 Multiplication

- Observation:
  - (a + b x) (b + c y) can be done with **three** multiplication, since bc + ad = **(a + b) (c + d)** - **ac** - **bd**.

$$x = \boxed{x_L}\boxed{x_R} = 2^{n/2}x_L + x_R$$

$$y = \boxed{y_L}\boxed{y_R} = 2^{n/2}y_L + y_R.$$

-

$$xy = (2^{n/2}x_L + x_R)(2^{n/2}y_L + y_R) = 2^n x_L y_L + 2^{n/2}(x_L y_R + x_R y_L) + x_R y_R.$$

-

- Multiplication strategy:
  - General: T(n) = 4 * T(n/2) + O(n) → O(**n^2**)
  - Reduced: T(n) = 3 * T(n/2) + O(n) → O(**$n^{1.59}$**)
  - Proof idea:
    - changes in the branching factor of recursion tree
    - Geometric increase from O(n) (k = 0) to O ($n^{\log_2 3}$) (k = $\log_2 n$).
    - DPK p. 52 - 53

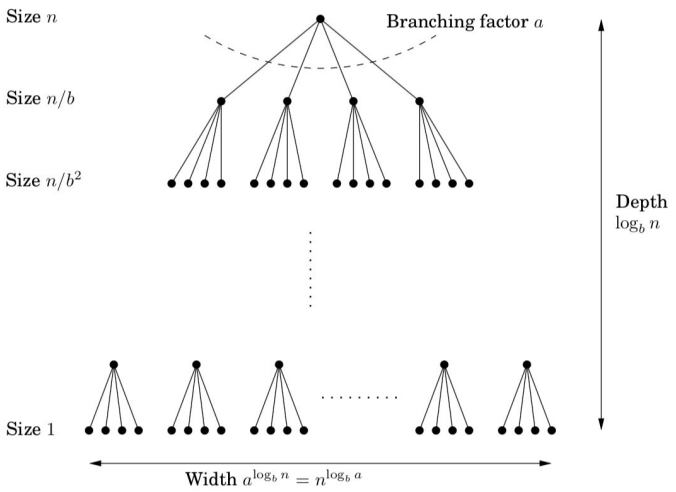
- K-terms
  - Split up number into k equally sized parts. Combine them with 2k - 1 multiplications instead of k^2.
  - $T(n) = (2k - 1)\, T(n/k) + O(n)$. → $T(n) = O(n^{\log(2k-1)/\log(k)})$

2. 2 Recurrence relationship

- Master Theorem:
  If $T(n) = aT([n/b]) + O(n^d)$ for some constants $a > 0$, $b > 1$ and $d \geq 0$. Then
  - $O(n^d)$ if $d > \log_b a$
  - $O(n^d \log n)$ if $d = \log_b a$
  - $O(n^{\log b\backslash a})$ if $d < \log_b a$

- Proof idea:

Figure 2.3 Each problem of size n is divided into a subproblems of size n/b.



-
  - K th level made up of a^k subproblems, each of size n / b^k. Total work for each level is

$$a^k \times O\left(\frac{n}{b^k}\right)^d = O(n^d) \times \left(\frac{a}{b^d}\right)^k.$$

  - A geometric series with **ratio $r = a / b^d$**
    - If $r < 1$, series decreasing, first term
    - If $r > 1$, sum is last term
    - If $r = 1$, all logn terms are equal to n^d

2.3 Merge Sort - All sorting algorithm that relies on comparisons takes n log(n).

- Algorithm: split into sub parts, recursively sort, and merge the list.

```
Function mergesort (a[1...n])
If n > 1:
      Return merge (mergesort(a[1...n/2]), mergesort(a[n/2]+1...n)
Else
      Return a

Function merge (x[1...k], y[1...l])
If k = 0: return y[1...l]
If l = 0: return x[1...k]
If x[1] ≤ y[1]
      Return x1 + merge(x[2...k], y[1...l])
Else
      Return y1 + merge(x[1...k], y[2...l])
```

- T(n) = 2 T(n / 2) + O(n) → **O(nlog(n))**

- Mergesort has the lower bound runtime for sorting. Consider the binary sorting tree. Every leaf is a permutation. Then there are n! Leafs. → n log(n)

- Proof: strong induction.

2.3.1 Quick sort

- ```
  Procedure quicksort(a[1..n])
  If n ≤ 1
        Return a
  Set v to be a random element in a
  Partition a into SL, Sv, SR
  Return quicksort(SL) SV quicksort(SR)
  ```

- Runtime: O(nlog(n)) expected runtime.
  ```

2.4 Median - all selection algorithm takes O(n)

- Sorting takes O(n log(n)) time, but we only care about the middle not the ordering.

- **Selection**
  - *Input*: list of numbers S, in integer k
  - *Output*: the kth smallest element of k

$$\text{selection}(S,k) = \begin{cases} \text{selection}(S_L,k) & \text{if } k \le |S_L| \\ v & \text{if } |S_L| < k \le |S_L| + |S_v| \\ \text{selection}(S_R, k - |S_L| - |S_v|) & \text{if } k > |S_L| + |S_v|. \end{cases}$$

-
- Shrink size of the sub-problem as max [S_L, S_R]
- If v is picked as the middle point, T(n) = T(n / 2) + O(n).

- **Efficiency Analysis**
  - Randomly choose v.
    - Best case: all mediums are picked. → O(n)
    - Worst case: pick in decreasing/increasing order → $O(n^2)$
  - Close to **best case**

- Prove by fair coin (p. 61-62) E = 1 + ½ E, E = 2. T(n) ≤ T(3n/4) + O(n).
- On average, expected in linear time **O(n)**.
- **Quicksort** takes O(n logn) on average, outperforms other sorting; use the same way to pick v as median to sort the array.

2.5 Matrix Multiplication

- Matrix multiplication computes n^2 cells, each take O(n). → $O(n^3)$.

- Break into subproblems, *blockwise*.

$$XY = \begin{bmatrix} A & B \\ C & D \end{bmatrix} \begin{bmatrix} E & F \\ G & H \end{bmatrix} = \begin{bmatrix} AE + BG & AF + BH \\ CE + DG & CF + DH \end{bmatrix}$$

-
- T(n) = 8 T(n / 2) + $O(n^2)$ → **$O(n^3)$**
- Improved by genius algebra:

$$XY = \begin{bmatrix} P_5 + P_4 - P_2 + P_6 & P_1 + P_2 \\ P_3 + P_4 & P_1 + P_5 - P_3 - P_7 \end{bmatrix}$$

where

$$\begin{aligned} P_1 &= A(F - H) & P_5 &= (A + D)(E + H) \\ P_2 &= (A + B)H & P_6 &= (B - D)(G + H) \\ P_3 &= (C + D)E & P_7 &= (A - C)(E + F) \\ P_4 &= D(G - E) \end{aligned}$$

- Reduced runtime to T(n) = **7** T(n / 2) + $O(n^2)$. → **$O(n^{2.81})$**

Lecture Notes

5- 17 Search, sort, select.

- Reduced and conquer
  - T(n) = aT(n - b)
  - If a > 1, takes exponential time.

- **Search**
  Input: Sorted list of integers; target integer.
  Output: index of the target.

  - Binary tree: log(n); Any search algorithm takes **O(log(n))**.
  - Degenerate into two parts, solve and combine.

- **Sort**

- List of sorting methods.
  Bubble sort, insertion sort, selection sort → **$O(n^2)$**
  Quicksort, mergesort → **O(nlog(n))**.

- Runtime:
  - N! Comparisons must be made
  - Traverse down the binary search tree with n! Leaves. → log(n!) < n log(n).
  - **O(nlog(n))**: best runtime for any sorting algorithm that relies on comparisons between elements.

5-22 DC examples

Power of two

- Given n, compute the digits of 2^n in decimal.

  - Cn digits.
  - ```
    Procedure PoT(n)
    If n = 0: return 1
    If n = 1: return 2
    P = PoT(n/2)                          // even: p = 2^(n/2);
                                          // odd: p = 2^((n-1)/2)
    P = KSMult(P, P)
    If n mod 2 = 1: P = add(P, P).
    Return P
    ```

  - Runtime: $T(n) = T(n/2) + O(n^{1.58})$

Making a binary heap

- Insert n elements, each take O(log(n)). In total takes O(nlog(n)).

- DC: put (o_1, k_1) aside, break remaining part into 2 halves. Make object 1 the root and tickle it down

- T(n) = 2 * T(n/2) + O(logn).

- Cheat MS: L(n) = 2L(n/2) + 1; U(n) = 2* U(n/2) + $n^{1/2}$; → T(n) = **O(n)**.

Greatest overlap
    ```

- Sort the list, and break into two part based on the median value.
- Get the greatest overlap on two sub problems.
- Get the greatest overlap between two subsets.

Minimum Distance

- Base:
    - If n = 2, return the distance
- Break into 2 halves of size n/2, (by x-value)
- Gives us the min distance on each side, d_L, d_R
- Compare $x - d_L \leq x_i \leq x + d_L$

2.6 Fast Fourier Transform

- Multiply two degree-d polynomials.
- Will not be on the exam.

- Polynomials:
    - $A(x) = a_0 + a_1x^1 + \ldots + a_{n-1}\, x^{n-1}$
    -