Ph.D. Qualifying Exam - Algorithms
Spring 2022
2 hours

- If any of the questions are not clear, please state your assumptions. If they are reasonable, they will be taken into consideration.
- Note that, credit will be given not to the length of your answers, but to their correctness and your grasp of the topic.
- There are six questions. The total grade is 100.

Question	**Max**	**Grade**
1	10	
2	10	
3	20	
4	15	
5	20	
6	25	
Total	100	

- **Question 1 (10 pts):**

  Indicate whether f=O(g) or f=$\Omega(g)$ or both (in which case f=$\theta(g)$) along with your justification.

  $f(n) = (\log n)^{\log n}$
  $g(n) = 2^{(\log n^2)}$

$2^{\log n^2} = 2^{2\log n} \sim 2^{\log n}$

$(\log n)^{\log n}$ will be slower than $2^{\log n}$

Hence $f(n) = O(g(n))$

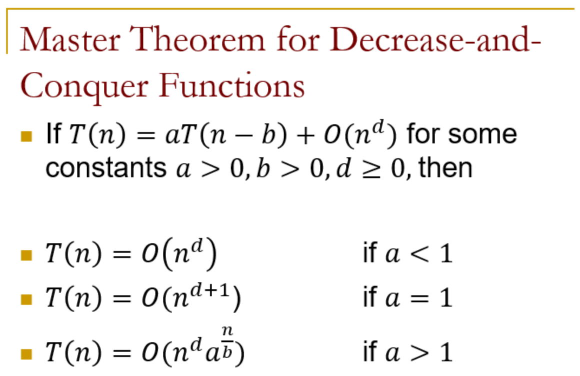
- **Question 2 (10 pts):**

  Arrange the following functions in the increasing order of complexity: (You can simply state the function names in the order you decide if you would like)

  $f1(n) = n\sqrt{5}$
  $f2(n) = n^5$
  $f3(n) = n!$
  $f4(n) = 5^n$
  $f5(n) = n \log n$

$f_1 < f_5 < f_2 < f_4 < f_3$

- **Question 3 (20 pts):**



Master Theorem for Decrease-and-Conquer Functions

- If $T(n) = aT(n-b) + O(n^d)$ for some constants $a > 0, b > 0, d \geq 0$, then

- $T(n) = O(n^d)$ if $a < 1$
- $T(n) = O(n^{d+1})$ if $a = 1$
- $T(n) = O(n^d a^{\frac{n}{b}})$ if $a > 1$

For the quicksort algorithm, choosing the left-most element as the pivot is a bad idea. 3a) (5pts) Explain why this is a bad idea. 3b) (5pts) What is the complexity of this worst-case scenario? 3c) (5pts) Justify your answer using the above Master Theorem. 3d) (5pts) Explain what are the values of a, b, and d.

3a) because each new subproblem is only reduced by 1

3b) $O(n^2)$

3c) and 3d) $a=1$, $b=1$, $d=1$ because of

Applying master theorem, you get $O(n^{1+1}) = O(n^2)$

- **Question 4 (15 pts):**

  Give one real-world scenario where Depth-first Search is used, and give one real-world scenario where Breadth-first Search is used. Describe what do nodes and edges represent in your examples, and why a DFS or BFS is helpful in your example.

DFS is used to find pre and post numbers in graphs which are often used in DAG linearization (scheduling problems) or for finding strongly connected components

E.g. : node = job/task
edge = connects a completed task to the next task

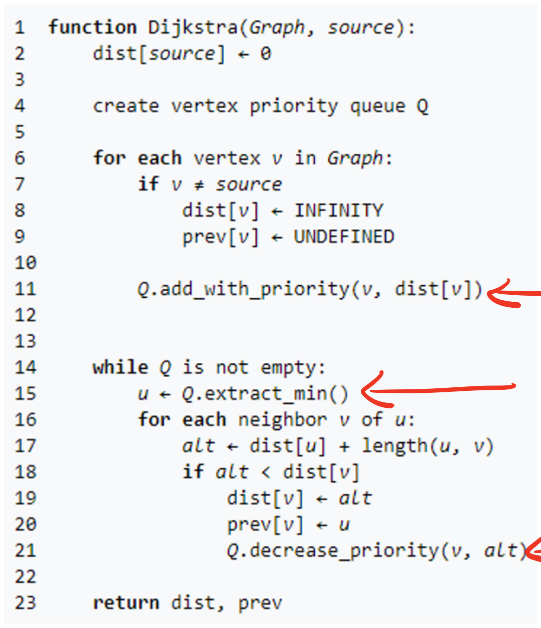
BFS is used to find shortest path between 2 nodes.

E.g. Road network
node = cities
edges = connection between 2 cities

- **Question 5 (20 pts):**

  5a) (5 pts) The following snippet of code is the pseudo-code for Djikstra's algorithm. What statements in the following code are the most dominant cost that contribute towards the time complexity for Djisktra's algorithm?

  5b) (15 pts) What is the complexity of Djisktra's if the underlying data structure used is an array instead of a binary heap? Show the breakdown of the complexity for the statements identified above, and then show the overall complexity.



```
1  function Dijkstra(Graph, source):
2      dist[source] ← 0
3
4      create vertex priority queue Q
5
6      for each vertex v in Graph:
7          if v ≠ source
8              dist[v] ← INFINITY
9              prev[v] ← UNDEFINED
10
11         Q.add_with_priority(v, dist[v])  ←
12
13
14     while Q is not empty:
15         u ← Q.extract_min()  ←
16         for each neighbor v of u:
17             alt ← dist[u] + length(u, v)
18             if alt < dist[v]
19                 dist[v] ← alt
20                 prev[v] ← u
21                 Q.decrease_priority(v, alt)  ←
22
23     return dist, prev
```

5a

5b) Line 15 called $O(V)$ times. Array implementation will take $O(V)$ for each

Lines 11 and 21 called $O(V+E)$ times. Array imp. will take $O(1)$ each time.

Hence total $= O(V)^2 + O(V+E)$

$= O(V^2)$

- **Question 6 (25 pts):**
  Dynamic Programming question: Given two strings s1 and s2, find the length of the longest subsequence in both the strings. The longest subsequence does not need to be contiguous.

  Input : s1 = A B C , s2 = B D C
  Output : 2
  Explanation : longest common subsequence is **B C**, whose length is 2

  Input : s1 = A B C B D A B, s2 = B D C A B A B
  Output : 5
  Explanation : longest common subsequence is **B C B A B**, whose length is 5

  Note: we solved longest common subsequence in class, which is different than the above. In the above question, the substring has to be contiguous.

  Q6a) (5pts) The naive solution for solving this problem includes enumerating all possible subsequences for both strings, and then finding the length of longest subsequence that is common in both strings. What is the complexity of the naive solution? Remember you have two strings here, so your complexity should consider 2 strings.

$O(2^{m+n})$

↳ 2^m for generating all subseq. for string 1

↳ 2^n for generating all subseq. for string 2

↳ Generating all combinations and choosing the best $2^m \times 2^n$

$= O(2^{m+n})$

Q6b (10pts) Fill the table

#	#	s	t	r	a	n	g
#	0	0	0	0	0	0	0
s	0	1	1	1	1	1	1
c	0	1	1	1	1	1	1
r	0	1	1	2	2	2	2
a	0	1	1	2	3	3	3
p	0	1	1	2	3	3	3

Q6c (15pts) What is the dynamic programming formulation for the following options?
i is the index for the row and j is the index for the column. *s1* and *s2* are the two strings.

– if s1[i] == s2[j]

$$dp[i][j] = 1 + dp[i-1][j-1]$$

– if s1[i] != s2[j]

$$\max(dp[i-1][j], dp[i][j-1])$$