December, 2021: Algorithms: Qualifier exam with solutions: closed book, closed notes, only allowable device (scientific calculator), Internet not allowed

1. (30 pts) Suppose you have n unsorted integers stored in an array.
   a. (15 pts) Write the algorithm (pseudocode) of merge sort for sorting n integers in nondecreasing order.

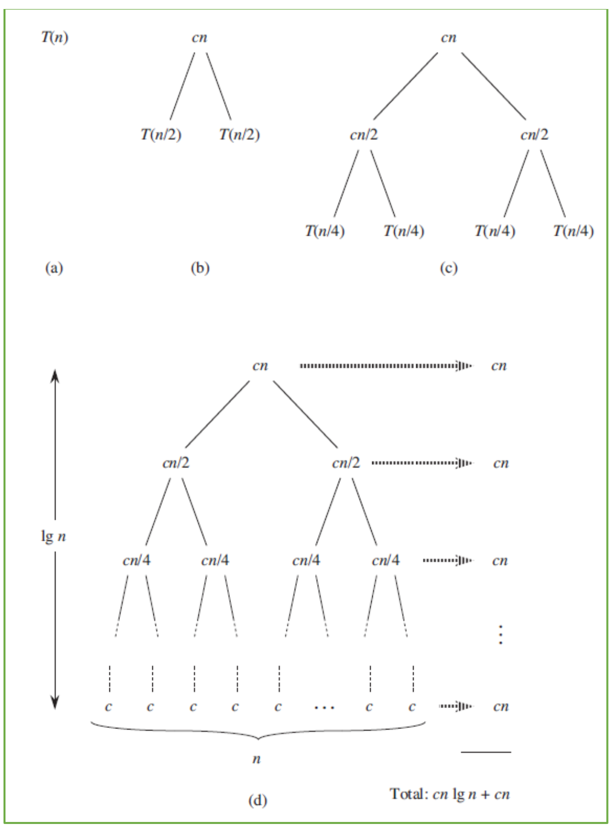
```
MERGE(A, p, q, r)
n1 = q - p + 1
n2 = r - q
let L[1 .. n1 + 1] and R[1 .. n2 + 1] be new arrays
for i = 1 to n1
    L[i] = A[p + i - 1]
for j = 1 to n2
    R[j] = A[q + j]
L[n1 + 1] = ∞
R[n2 + 1] = ∞
i = 1
j = 1
for k = p to r
    if L[i] ≤ R[j]
        A[k] = L[i]
        i = i + 1
    else A[k] = R[j]
        j = j + 1
```

```
MERGE-SORT(A, p, r)
if p < r
    q = ⌊(p + r)/2⌋
    MERGE-SORT(A, p, q)
    MERGE-SORT(A, q + 1, r)
    MERGE(A, p, q, r)
```

   b. (5 pts) Write the recurrence relation of merge sort in the form of $T(n)$.

$$T(n) = \begin{cases} \Theta(1) & \text{if } n = 1\,, \\ 2T(n/2) + \Theta(n) & \text{if } n > 1\,. \end{cases}$$

   c. (10 pts) Derive the worst-case runtime complexity of merge sort in big-Θ notation by visualizing recurrence tree.



- Total number of levels = $log_2 n + 1$
- At each level, the associated cost cn
- Total cost = $cn(log_2 n + 1)$
- Therefore, total cost is $\Theta(nlog_2 n)$

2. (15 pts) Estimate big-O of the following function and algorithm.

   a. (10 pts) Estimate big-O of the function $f(n) = 7nlog(n!) + (n^2 + 5)logn$. Show your calculations.

   Step 1

   $7n$ is $O(n)$

   $n! = 1 * 2 * 3 * \ldots * n \leq \mathrm{n} * \mathrm{n} * \mathrm{n} * \ldots * \mathrm{n} = \mathrm{n}^{\mathrm{n}}$

   $\log(n!) \leq \log(n^n) = nlogn$

   $\log(n!)$ is $O(nlogn)$

   $7nlog(n!)$ is $O(n^2 logn)$

   Step 2:

   $(n^2 + 5)$ is $O(n^2)$

   $(n^2 + 5)logn$ is $O(n^2 logn)$

   Step 3

   $f(n) = 7nlog(n!) + (n^2 + 5)logn$ is $O(\max(n^2 logn,\ n^2 logn))$

   $f(n) = 7nlog(n!) + (n^2 + 5)logn$ is $O(n^2 logn)$

   b. (5 pts) Estimate worst case runtime complexity in big-O for the following function. The function computes nth Fibonacci number.

```
int nthFibonacci(int n){
    if(n <= 0)
        return 0;
    else if(n == 1)
        return 1;
    return nthFibonacci(n-1) + nthFibonacci(n-2);
}
```

   Big-O estimate of the recursive functions: $O\left(\#branches^{depth}\right)$. There are 2 branches per call, and we go as deep as n. Therefore, the worst case runtime is $O(2^n)$

3. (20 pts) Write the runtime complexity in big-Θ notation for the following recurrence relations using Master theorem. The Master theorem is as follows. Show your calculations.

Theorem 4.1 (Master theorem)

Let $a \geq 1$ and $b > 1$ be constants, let $f(n)$ be a function, and let $T(n)$ be defined on the nonnegative integers by the recurrence

$$T(n) = aT(n/b) + f(n),$$

where we interpret n/b to mean either $\lfloor n/b \rfloor$ or $\lceil n/b \rceil$. Then $T(n)$ has the following asymptotic bounds:

1. If $f(n) = O(n^{\log_b a - \epsilon})$ for some constant $\epsilon > 0$, then $T(n) = \Theta(n^{\log_b a})$.
2. If $f(n) = \Theta(n^{\log_b a})$, then $T(n) = \Theta(n^{\log_b a} \lg n)$.
3. If $f(n) = \Omega(n^{\log_b a + \epsilon})$ for some constant $\epsilon > 0$, and if $af(n/b) \leq cf(n)$ for some constant $c < 1$ and all sufficiently large n, then $T(n) = \Theta(f(n))$. ■

   a. (10 pts) $T(n) = 8T(n/2) + \Theta(n^2)$

   $a = 8,\ b = 2,\quad n^{\log_2 8} = n^3$

   $n^2 = O(n^3)$ → case 1

   $T(n) = \Theta(\mathrm{n}^3)$

b. (10 pts) $T(n) = 3T\left(\frac{n}{4}\right) + nlogn$

$a = 3,\ b = 4,\quad n^{\log_4 3} = n^{0.793}$

We have $f(n) =$ nlogn. Let's compare it with $n^{0.8}$

Let's assume $\in = 0.2$

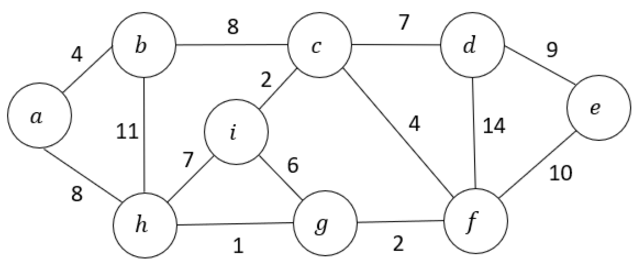
So, $n^{0.8+0.2} = n$

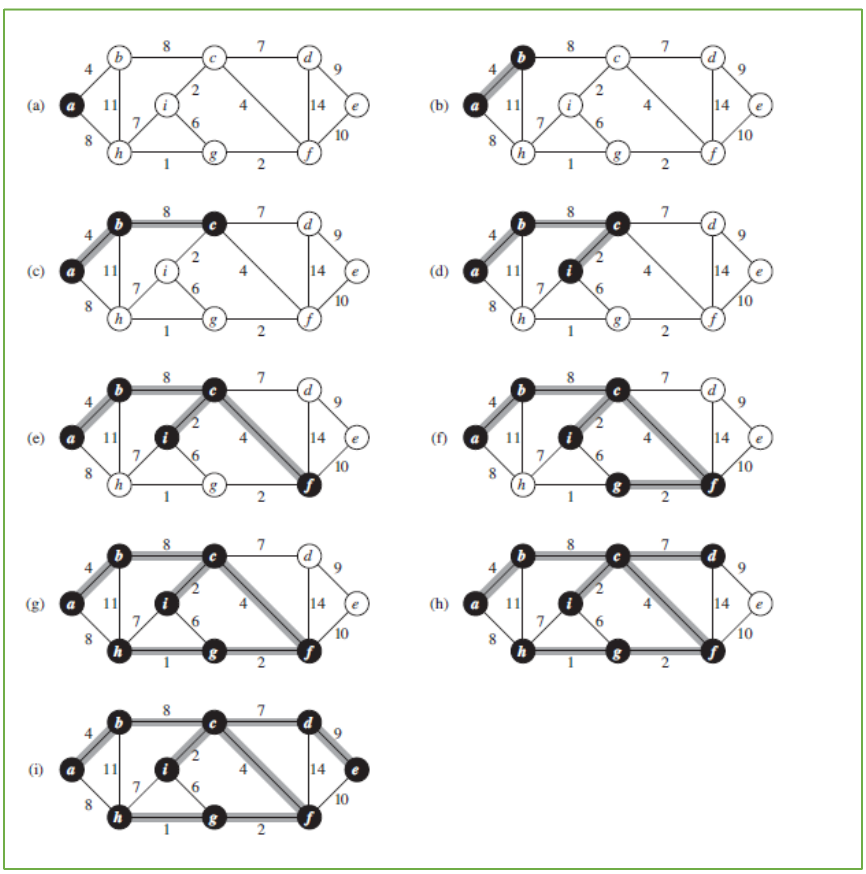
$nlogn = \Omega(n)$

Moreover, $af\left(\frac{n}{b}\right) = 3f\left(\frac{n}{4}\right) = 3\left(\frac{n}{4}\right)\log\left(\frac{n}{4}\right) \leq \frac{3}{4}\text{nlog}\,n$ for $c = \frac{3}{4}$

Applying case 3, $T(n) = \Theta(\text{nlog}\,n)$

4. (15 pts) Compute minimum spanning tree by Prim's algorithm from the following undirected graph. Consider the starting node is a. Show intermediate sequential tree building steps. List the final tree edges and write the final tree weight.





Sequentially select the light edges that cross the node partitions (*tree vs not yet in tree*)

5. (20 pts) Suppose we have two DNA sequences $X = "ACGCTAC"$ and $Y = "CTGACA"$.
    a. (15 pts) Use dynamic programming to find the length of the longest common subsequence of X and Y.

	j	0	1	2	3	4	5	6
i		y_j	C	T	G	A	C	A
0	x_i	0	0	0	0	0	0	0
1	A	0	↑ 0	↑ 0	↑ 0	↖ 1	← 1	↖ 1
2	C	0	↖ 1	← 1	← 1	↑ 1	↖ 2	← 2
3	G	0	↑ 1	↑ 1	↖ 2	← 2	↑ 2	↑ 2
4	C	0	↖ 1	↑ 1	↑ 2	↑ 2	↖ 3	← 3
5	T	0	↑ 1	↖ 2	↑ 2	↑ 2	↑ 3	↑ 3
6	A	0	↑ 1	↑ 2	↑ 2	↖ 3	↑ 3	↖ 4
7	C	0	↖ 1	↑ 2	↑ 2	↑ 3	↖ 4	↑ 4

Length of LCS = 4

    b. (5 pts) Write the longest common subsequence Z.
    Bottom-up traversal of the table; diagonal arrow encounter; Z = “CGCA”