CPSC 320 2023S2: Quiz 3 Solutions

1 Multi-way merges (9 points)

You need to efficiently merge $k \geq 1$ sorted arrays, each containing $n \geq 1$ elements, to get a sorted array with kn elements. We'll denote the k arrays by X_1, X_2, ... X_k; here each X_i corresponds to $\mathrm{X}_i[1..n]$ since it contains n elements. Merging the sorted arrays produces a final sorted "multi-way merge" result with nk elements. For example, if the arrays are X_1 = [2,4,7], X_2 = [1,3,9] and X_3 = [3,5,6] then $n = k = 3$ and the result is [1,2,3,3,4,5,6,7,9].

Assume that you already have a function 2-Way-Merge(A[1..n],B[1..m]) that takes two sorted arrays, possibly of different sizes n and m, and returns the sorted array of size $n + m$ obtained by merging A and B. The runtime of 2-Way-Merge(A[1..n],B[1..m]) is $\Theta(n + m)$.

Using the 2-Way-Merge function, an iterative multi-way merge algorithm is as follows:

procedure ITERATIVE-MERGE($X_1, X_2, \ldots, X_k$)
  ▷ $k \geq 1$ and $n \geq 1$; each X_i is an array of n elements
  create a new array Y_1 with n elements
  $Y_1 \leftarrow X_1$
  for i from 2 to k **do**
    create a new array Y_i with in elements
    $Y_i \leftarrow$ 2-Way-Merge(X_i, Y_{i-1})
  return Y_k

1. What is the asymptotic running time of Iterative-Merge? Choose one option.

○ $\Theta(nk)$ ○ $\Theta(nk \log k)$ ○ $\Theta(n^2k)$ ● $\Theta(nk^2)$ ○ $\Theta(n^2k^2)$

2. Another approach to multi-way merge is divide and conquer (DC):

```
procedure DC-MERGE(X_1, X_2, ..., X_k)
    ▷ k ≥ 1 and n ≥ 1; each X_i is an array of n elements
    if k == 1 then return X_1                          ▷ Assume that this takes Θ(n) time
    else                                                                        ▷ k ≥ 2
        create a new array Y with (⌊k/2⌋)n elements     ▷ Assume this takes O(1) time
        create a new array Y' with (k − ⌊k/2⌋)n elements ▷ Assume this takes O(1) time
        Y ← DC-Merge(X_1, ..., X_⌊k/2⌋)
        Y' ← DC-Merge(X_⌊k/2⌋+1, ..., X_k)
        return 2-Way-Merge(Y, Y')
```

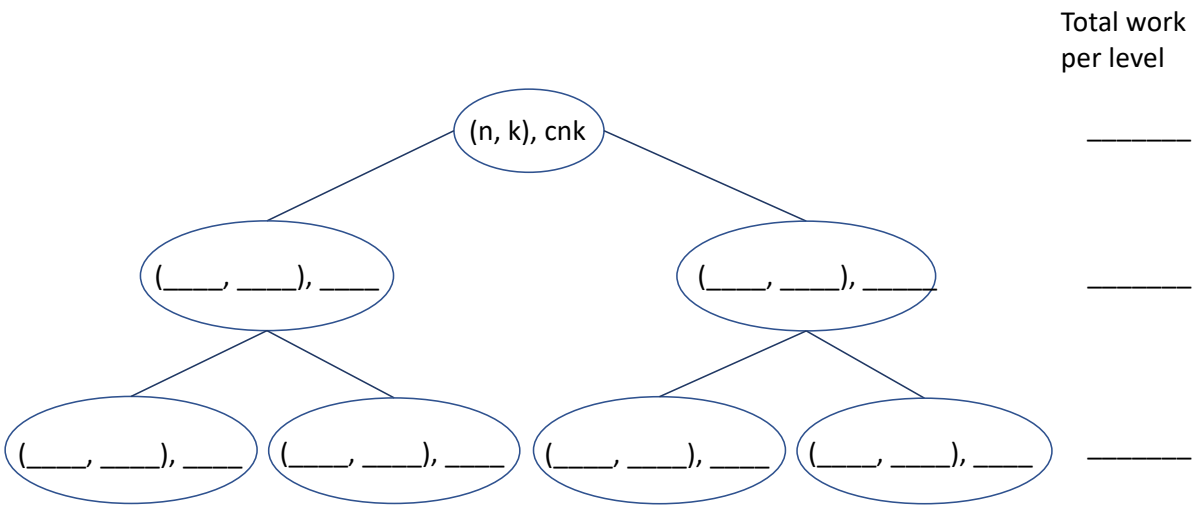
Let $T(n, k)$ denote the running time of DC-Merge. Complete the following recurrence for $T(n, k)$. You can ignore floors and ceilings.

SOLUTION: There are two subproblems with roughly half the number of arrays to merge. Both arrays have roughly equal size, with a number of entries proportional to nk. So the time to do the 2-way merge is $\Theta(nk)$. We have that for some constant $c > 0$,

$$T(n,k) = \begin{cases} cn, & \text{if } k = 1 \\ 2T(n,k/2) + cnk, & \text{otherwise.} \end{cases}$$

Solving we get that $T(k) = 2^{\log_2 k} T(1) + cn \sum_{i=1}^{\log_2 k} k = \Theta(nk \log k) = \Theta(n^2 \log n)$, when $k = n$.

3. Here is an incomplete recursion tree for DC-Merge. Each node of the tree corresponds to a recursive call and is labelled with three numbers. The first two numbers describe the size of the subproblem, and the third bounds the time for the work done at the node (not counting subsequent recursive calls). The time is expressed using a positive constant c. *You do not need to fill in the missing entries in the tree, but you can if it helps you answer the following questions.*



What is the label of the leftmost child of the root? Choose one.

○ $(n, k), cnk/2$ ○ $(n/2, k), cnk/2$ ● $(n, k/2), cnk/2$

○ $(n, k), cnk$ ○ $(n/2, k), cnk$ ○ $(n, k/2), cnk$

4. What is the total work at level 0 (the root)?

○ $cnk/2$ ● cnk ○ $2cnk$

5. What is the total work at level 1?

○ $cnk/2$ ● cnk ○ $2cnk$

6. How many levels does the full tree (not shown) have? Choose one option.

○ $\Theta(\log n)$ ● $\Theta(\log k)$ ○ $\Theta(\log(nk))$

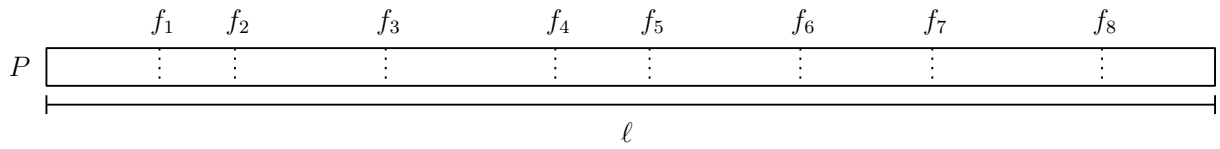
7. What is the asymptotic running time of Algorithm DC-Multi-Way-Merge? Choose one option.

○ $\Theta(nk)$ ● $\Theta(nk \log k)$ ○ $\Theta(n^2 k)$ ○ $\Theta(nk^2)$ ○ $\Theta(n^2 k^2)$

2 Making the Cut (5 points)

A rectangular strip of paper $\mathcal{P}$ of length ℓ has n vertical folds, located at distances $f_1, f_2, \ldots, f_n$ from the left end of $\mathcal{P}$. We'll let $f_0 = 0$ and $f_{n+1} = l$, and assume that all of the f_i are distinct. See the figure below, where $n = 8$.

For a given positive M, you want to cut $\mathcal{P}$ along a minimum number of folds so that the resulting strips $\{s_1, \ldots, s_k\}$ each have length at most M. You can assume that $M \geq f_{i+1} - f_i$ for any $i, 0 \leq i \leq n$.



Here is a greedy algorithm that correctly solves the problem in $O(n)$ time, when an instance is the *sorted* list $f_1, f_2, \ldots, f_n$, plus l and M:

> Sweep P from left to right, to find the fold that is as far right as possible but still within distance M from the left end of P. Cut off the strip and repeat until the right end of P is reached.

1. Let $s_1^g, \ldots, s_{k_g}^g$ denote the lengths of the strips, from left to right, that this algorithm produces.
   Let $s_1^*, \ldots, s_{k_*}^*$ denote the lengths of the strips, from left to right, of some optimal solution.
   A "greedy stays ahead" correctness argument compares these two sequences. Which of the following inequalities **must** hold? Check all that apply.
   - ○ $s_i^* \leq s_i^g$ for all $i, 1 \leq i \leq k^*$
   - ○ $s_i^g \leq s_i^*$ for all $i, 1 \leq i \leq k^*$
   - ● $k_* \leq k_g$
   - ● $k_g \leq k_*$

2. An alternative greedy algorithm sweeps P from *right to left*, to find the fold that is as far *left* as possible, but still within distance M from the right end of P. Cut off the strip and repeat until the left end of P is reached.
   Is this a correct algorithm?
   ● Yes ○ No

3. Yet another greedy algorithm (that can take advantage of two people doing the cutting) is to simultaneously run the "sweep-left" and "sweep-right" algorithm, repeatedly cutting one piece off each side, until the piece that is left in the middle has length at most M. That is, a piece is cut off the left according to the sweep-left algorithm, and a piece is cut off the right according to the sweep-right algorithm, and this is repeated until the remaining middle piece has length at most M.
   Is this a correct algorithm?
   ● Yes ○ No
   [Here's a correctness argument. Let C be the set of cuts made by the "sweep-right" part of this algorithm, say ending at fold f (or the left end of the paper strip). C is a subset of at least one optimal solution, namely that produced by the algorithm of part 1. In fact, we can take the union of C with *any* optimal solution for the subregion from fold f to the right end. The set C' of cuts made by the "sweep-left" part of the algorithm must be optimal for this subregion, since sweep-left is optimal by part 2. So $C \cup C'$ is an optimal solution, and is the one produced here.]

3 Lost in Nirvana (5 points)

On one of her swimming adventures, Anne swam out further than usual in an beautiful but unfamiliar lake. Having paid no attention to landmarks around the lake, she swam back to shore in a somewhat different direction than on the way out. Once back on shore, she had no idea whether her belongings were off to her left or off to her right. What strategy should she use to find them? She wants to minimize distance walked since she's already shivering from staying too long in the water, and the gravel trail around the lake isn't kind to her feet. The lake is huge and she doesn't want to walk the full perimeter (it might be helpful to think of the length of the perimeter as being infinite).

Here is one option that she could try, from her exit point from the lake:

```
done ← false
i ← 1
while not done do
    walk i steps to the left
    if belongings are reached while walking then
        done ← true
    else
        return to exit point
        walk i steps to the right
        if belongings are reached while walking then
            done ← true
        else
            return to exit point
    i ← i + 1
```

1. Let n be the direct distance (measured as number of steps) from Anne's exit spot to her belongings. As a function of n, how many steps does Anne take if she uses the above algorithm? Choose one.

   ○ $\Theta(n)$ ○ $\Theta(n \log n)$ ● $\Theta(n^2)$

2. The algorithm is repeated below. Make a small modification to *one line of* the algorithm that improves the runtime. You can circle the part you want to modify and write the modified code on the right. (Keep in mind that just going around the perimeter of the lake is not viable. Also, since Anne doesn't know the distance n to her belongings, the code that you write should not refer to n, although its runtime does depend on n.)

```
done ← false
i ← 1
while not done do
    walk i steps to the left
    if belongings are reached while walking then
        done ← true
    else
        return to exit point
        walk i steps to the right
        if belongings are reached while walking then
            done ← true
        else
            return to exit point
    i ← 2i          ▷ this is the changed line
```

3. How many steps does your algorithm of part 2 take? Choose one. [1]

● $\Theta(n)$ ○ $\Theta(n \log n)$ ○ $\Theta(n^2)$

[1] Hint: Depending on the change you made to the algorithm, the geometric sum formula (from the Divide and Conquer worksheet) might be useful to you here. A *geometric sum* has the form $\sum_{i=0}^{n} x^i$, where $x > 0$. When $x \neq 1$ we have that:

$$\sum_{i=0}^{n} x^i = \frac{x^{n+1} - 1}{x - 1} = \frac{1 - x^{n+1}}{1 - x}.$$