Weekly Assignment 3
Total: 100

CS 2500: Algorithms

Due Date: October 7, 2024 at 11.59 PM

Instructions

- Submit your solutions by the deadline specified above.
- Ensure that your work is your own.
- Write your answers clearly and show all your work.
- If you have any questions, ask during recitations or office hours.

1 Problems

1.1 Max Heap Operation [12 points]

1. Illustrate the operation of `Max-Heapify(A,3)` on the array `A = [27,17,3,16,13,10,1,5,7,12,4,8,9,0`. [**4 points**]
2. Starting with the algorithm for `Max-Heapify`, write the algorithm for `Min-Heapify(A,i)`, which performs the corresponding manipulation on a minheap. How does the running time of `Min-Heapify` compare with that of `Max-Heapify`? [**3 points**]
3. What is the effect of calling `Max-Heapify(A,i)` when the element `A[i]` is larger than its children? [**2 points**]
4. What is the effect of calling `Max-Heapify(A,i)` for $i >$ A.heap-size/2? [**3 points**]

1.2 Build Heap Operation [14 points]

1. Prove that algorithm `Build-Max-Heap` given on Slide 34 of Lecture 12 is correct using the loop invariant method. [**4 points**]
2. Illustrate the operation of `Build-Max-Heap` on the array `A = [5,3,17,10,84,19,6,22,9]`. [**3 points**]
3. We obtained a tighter asymptotic bound on `Build-Max-Heap` in class. In doing so, we assumed the following:

   (a) There are at most $\lceil \frac{n}{2^{h+1}} \rceil$ elements with a height h in a heap with n elements. [**3 points**]

   (b) $\lceil \frac{n}{2^{h+1}} \rceil \geq \frac{1}{2} \quad \forall h \in [0, \lfloor \log n \rfloor]$ [**4 points**]

   Prove (a) and (b) above.

1.3 Heap Sort [12 points]

1. Illustrate the operation of `Heap-Sort` on the array `A = [5,13,2,25,7,17,20,8,4]`. **[3 points]**
2. Argue the correctness of `Heap-Sort` using the following loop invariant **[4 points]**

   > At the start of each iteration of the for loop of lines 2-5, the subarray `A[1:i]` is a max-heap containing the i smallest elements of `A[1:n]`, and the subarray `A[i+1:n]` contains the $n - i$ largest elements of `A[1:n]`, sorted.

3. What is the running time of `Heap-Sort` on an array A of length n that is already sorted in increasing order? How about if the array is already sorted in decreasing order? **[5 points]**

1.4 Merge Sort [12 points]

1. Suppose $a[1 : m]$ and $b[1 : n]$ both contain sorted elements in non-decreasing order. Write an algorithm that merges these items into $c[1 : m + n]$. Your algorithm should be shorter than algorithm Merge (Slide 10, Lecture 11) since you can now place a large value in $a[m + 1]$ and $b[n + 1]$. **[4 points]**
2. The sequences $X_1, X_2, \ldots, X_\ell$ are sorted sequences such that $\sum_{i=1}^{\ell} |X_i| = n$. Show how to merge these ℓ sequences in time $O(n \log \ell)$. **[8 points]**

2 Finding the Median of Two Sorted Arrays

This problem tests your ability to develop efficient algorithms using the divide-and-conquer approach. It's a popular interview question at leading tech companies like Google, Facebook, and Amazon, where it's used to assess your problem-solving skills, particularly in software engineering and data science roles.

- You must submit both the source code and a brief report that explains your approach, the algorithms used, and any challenges encountered.
- Ensure your code is well-documented, with comments explaining the purpose of key sections and functions.
- Your code will be evaluated on both correctness and efficiency, as well as clarity and organization.

2.1 Coding Exercise [20 points]

2.1.1 Task

Write a function `findMedianSortedArrays(nums1, nums2)` that takes two sorted arrays, `nums1` and `nums2` and returns the median of the two arrays. The function must run in $O(\log(m + n))$ time.

2.1.2 Constraints

- `nums1.length = m`
- `nums2.length = n`
- $0 \leq m \leq 1000$
- $0 \leq n \leq 1000$
- $1 \leq m + n \leq 2000$
- $-10^6 \leq nums1[i], nums2[i] \leq 10^6$

2.1.3 Examples

- **Example 1:**
  - **Input:** `nums1 = [1,3], nums2 = [2]`
  - **Output:** `2.00000`
  - **Explanation:**
    The merged array is `[1,2,3]`, and the median is `2`.
- **Example 2:**
  - **Input:** `nums1 = [1,2], nums2 = [3,4]`
  - **Output:** `2.50000`
  - **Explanation:**
    The merged array is `[1,2,3,4]`, and the median is $\frac{2+3}{2} = 2.5$.

2.2 Analysis [30 points]

1. Analyze the time complexity of your implementation. **[5 points]**
2. What is the space complexity of your solution? Discuss whether there are trade-offs between time and space complexity in your approach. **[5 points]**
3. How does your solution ensure that the median is found in $O(\log(m+n))$ time? Describe the key steps and techniques used to achieve this efficiency. **[5 points]**
4. Describe how your algorithm handles the following edge cases:
   (a) $m = 0$, $n = 1$ (i.e., one of the arrays is empty). **[2 points]**
   (b) Arrays with equal elements (e.g., $nums1 = [2, 2], nums2 = [2, 2]$). **[2 points]**
   (c) Arrays of drastically different sizes (e.g., $nums1 = [1], nums2 = [10^6 - 1, 10^6]$). **[4 points]**
5. How does your algorithm perform as the values of m and n approach their limits (e.g., $m + n = 2000$)? What are the potential bottlenecks, and how might you address them? **[4 points]**
6. Suppose the problem was extended to find the median of three sorted arrays, $nums1$, $nums2$, and $nums3$. How would your approach change to handle this new constraint? **[3 points]**

2.3 Bonus Question: Optimization Challenge [10 points]

For very large inputs, your algorithm should minimize unnecessary operations. Explain and implement any optimizations (e.g., eliminating unnecessary comparisons or handling special cases) that can further improve your algorithm's performance in practice.