Algorithm Midterm Rehearsal

Saiyang

Oct 13, 2023

Outline

Midterm Exam

Time	*<Fri, Oct 20, 2023>*, 1:00 p.m. to 3:50 p.m.
Covered Contents	Chapter 1-4, 6-9, Chapter 12-13, Chapter 30

Exam Format

- True or False (10 Problems, 10 points, 1 point each)
- Multiple Choice (10 Problems, 30 points, 3 point each)
- Open Questions (5~7 Problems, 60 points)

How to Review

- Questions? (60%+ from homeworks)
- Homeworks
- Lectures
- Textbooks

Practice – True or False

- Binary insertion sorting (insertion sort that uses binary search to nd each insertion point) requires O(n log n) total operations.

Practice – True or False

- Binary insertion sorting (insertion sort that uses binary search to nd each insertion point) requires O(n log n) total operations.
  - Solution: False.
  - While binary insertion sorting improves the time it takes to find the right position for the next element being inserted, it may still take O(n) time to perform the swaps necessary to shift it into place. This results in an O(n2) running time, the same as that of insertion sort.

Practice – True or False

- In a BST, we can find the next smallest element to a given element in O(1) time.

Practice – True or False

- In a BST, we can find the next smallest element to a given element in O(1) time.
  - Solution: False.
  - Finding the next smallest element, the predecessor, may require traveling down the height of the tree, making the running time O(h).

Practice – Multiple Choices

- You are running a library catalog. You know that the books in your collection are almost in sorted ascending order by title, with the exception of one book which is in the wrong place. You want the catalog to be completely sorted in ascending order.

a Insertion Sort

b Merge Sort

c Radix Sort

d Heap Sort

e Counting Sort

Practice – Multiple Choices

- You are running a library catalog. You know that the books in your collection are almost in sorted ascending order by title, with the exception of one book which is in the wrong place. You want the catalog to be completely sorted in ascending order.

a Insertion Sort

b Merge Sort

c Radix Sort

d Heap Sort

e Counting Sort

- Solution: [a]

Practice – Multiple Choices

- Which of the following complexity analysis is/are correct.

a The worst case running time for building a binary search tree is O(n lg n).

b The worst case running time for building a red-back tree is O(n lg n).

c The worst case running time for building a binary search tree is O(n2).

d The worst case running time for building a red-back tree is O(n2).

- Solution: [b,c]

Practice – Questions

Give asymptotic upper and lower bound for $T(n)$ in each of the following recurrences. Assume that $T(n)$ is constant for $n \leqslant 2$. Make your bounds as tight as possible, and justify your answers.

- $T(n) = 2\,T(n/2) + n^4$
- $T(n) = T(n-2) + n^2$

Practice – Question I

Given the recurrence:

$$T(n) = 2\,T\left(\frac{n}{2}\right) + n^4$$

Using the Master Theorem, we determine that the function $f(n) = n^4$ grows faster than $n^{\log_b a} = n$, where $a = 2$ and $b = 2$.
For the Master theorem's case 3 to apply, two conditions must be satisfied:

1. $f(n) = \Omega(n^{\log_b a+\epsilon})$ for some $\epsilon > 0$. In our case, $f(n) = n^4$ which is $\Omega(n^{1+\epsilon})$ where $\epsilon = 3$.
2. $af(n/b) \leq kf(n)$ for some $k < 1$ and sufficiently large n.

Both conditions are met, thus by the Master theorem (Case 3), the solution is:

$$T(n) = \Theta(n^4)$$

Practice – Question

Consider the recurrence:

$$T(n) = T(n-2) + n^2$$

Expanding this recurrence iteratively, we have:

$$\begin{aligned} T(n) &= T(n-2) + n^2 \\ T(n-2) &= T(n-4) + (n-2)^2 \\ &\vdots \\ T(n) &= T(0) + 2^2 + 4^2 + \cdots + (n-2)^2 + n^2 \end{aligned}$$

Assuming $T(0)$ is some constant c, the sum of the squares up to n grows as $O(n^3)$. Thus, we can conclude:

$$T(n) = O(n^3)$$

Divide and conquer

- Use when you can partition problems into unrelated subproblems.
- Key step: How to merge the solutions.
- Analysis:
  - Use recursion trees for running time (or guess and do induction)
  - Usually use induction to prove correctness.

Dynamic Programming

- Use if all subproblems are highly related, and there are not many subproblems you need to solve.
- Define a table for your subproblems.
- Consider all possibilities in the last step, write the recursion function for the table.
- Usually use induction to prove correctness (correctness is often straightforward and you only need to go through the formalities)
- Try implement the basic algorithms we covered in class.

Sorting

What you need to familiar with?

- Design algorithm
- Complexity / running time
    - Time / space
    - Worse case / average case / best case
- Proof of correctness

Sorting

- Insertion sort
- Merge sort
- Heap sort
- Quick sort
- Counting sort
- Radix Sort

Insertion sort

- Basic idea
  - builds the final sorted array (or list) one item at a time
- $O(n^2)$ worst case
- $O(n^2)$ average case

Merge sort

- Basic idea
  - It divides the input array into two halves, calls itself for the two halves, and then merges the two sorted halves.
- $O(n \log n)$ worst case

Heap sort

- Uses the very useful heap data structure
  - Complete binary tree
  - Heap property: parent key $>$ children's keys
- $O(n \log n)$ worst case
- Sorts in place
- Fair amount of shuffling memory around

Quick sort

- Divide-and-conquer:
  - Partition array into two subarrays,
- recursively sort
  - All of first subarray $<$ all of second subarray
  - No merge step needed!
- $O(n \log n)$ average case
- Fast in practice
- $O(n^2)$ worst case
  - Naïve implementation: worst case on sorted input
  - Address this with randomized quicksort

Count sort

- Basic idea
  - Works by counting the number of objects having distinct key values
- Running time complexity is $O(n)$ with space proportional to the range of data.

Radix sort

- Basic idea
    - do digit sort starting from least significant digit to most significant digit. Radix sort uses counting sort as a subroutine to sort.
- Running time is $O(n \log n)$

Binary Search Trees

- Basic algorithm:
  - Insert elements of unsorted array from $1..n$
  - Do an inorder tree walk to print in sorted order
- Running time
  - Best case: $(n \log n)$ (it's a comparison sort)
  - Worst case: $O(n^2)$
  - Average case: $O(n \log n)$ (it's a quicksort)

Red-Black Tree

- Red-black trees:
    - Binary search tree with an additional attribute for its nodes: color which can be red or black
    - “Balanced” binary search trees guarantee an $O(lgn)$ running time
- properties:
    - Every node is either red or black
    - Every leaf (NULL pointer) is black - every “real” node has 2 children
    - If a node is red, both children are black - can’t have 2 consecutive reds on a path
    - Every path from node to descendent leaf contains the same number of black nodes
    - The root is always black

Exam Tips

- It would be quite obvious what technique you should use.
- There will be an algorithm design question for a sorting algorithm but there is no need to write code.
- Problems are not necessarily sorted in level of difficulty.

Q & A

Good Luck!