Review

Ari Mermelstein

March 2019

1 Review Material

1.1 Discrete math

1. Induction
2. Sets
3. Logic

1.2 Calculus

1. Limits
2. L'Hôpital's rule
3. derivatives

1.3 Data structures

1. Recursion
2. Linked lists
3. stacks
4. queues
5. BSTs

2 Binary search

```
Input: An array A[1..n]
Input: An integer n
Input: An integer to search for called target
Output: An index 1 ≤ i ≤ n such that A[i] = target
start = 1;
end = n;
while start ≤ end do
    mid = ⌊(start+end)/2⌋;
    if target = A[mid] then
        return mid;
    end
    else if target < A[mid] then
        end = mid-1;
    end
    else
        start = mid+1;
    end
end
return NOT_FOUND;
```

2.1 How long does binary search take?

Let $T(n)$ be the worst case running time for binary search on an array of size n. Then

$$T(n) = T(\frac{n}{2}) + 1$$

is an equation that represents the running time.
How do we solve this? One way is by substitution.

2.2 Substitution method

1. Guess a solution
2. Prove it's correct by induction.

$T(n) = T(\frac{n}{2}) + 1$

$T(\frac{n}{2}) = T(\frac{n}{4}) + 1$

so $T(n) = T(\frac{n}{4}) + 1 + 1$

...

$T(n) = T(\frac{n}{2^k}) + k$

We Want $2^k = n \Rightarrow k = \log_2 n$.

$T(n) = 1 + \log_2 n$.

2. Prove by induction.
We want to prove that $T(n) \leq c \log_2 n$ for some positive c and for all n.

Base case: $T(k) = \log_2 k \leq c$ as long as c is sufficiently large.

Inductive case:

$T(n) = T(\frac{n}{2}) + 1$

By the inductive hypothesis

$\Rightarrow T(n) \leq c \log_2(\frac{n}{2}) + 1$

$\Rightarrow T(n) \leq c(\log_2 n - \log_2 2) + 1$

$\Rightarrow T(n) \leq c \log_2 n - c + 1$

$\Rightarrow T(n) \leq c \log_2 n - (c + 1)$

$\Rightarrow T(n) \leq c \log_2 n$ as long as $c + 1 > 0 \Rightarrow c \geq 1$

2.3 Questions similar to binary search

3 Sorting

3.1 InsertionSort pseudocode

```
Input: An array A[1..n]
Input: An integer n
for i=2 to n do
    key = A[i];
    location = i-1 ; // location represents the locations we are looking at to see
      where the key goes
    while location > 0 and A[location] > key do
        A[location+1] = A[location];
        location = location-1;
    end
    A[location + 1] = key;
end
```

3.2 Analyzing insertion sort

The outer loop happens $n-1$ times. The inner loop, in the worst case has to run $i-1$ steps. So the worst case time is given by:

$$\sum_{j=1}^{n-1} j = \frac{n(n-1)}{2} = \Theta(n^2)$$

3.3 MergeSort Pseudocode

3.4 Merge

```
Input: An array A
Input: integers start1, end1, start2, end2
; // We will view A as having 2 subarrays A[start1..end1] and
  A[start2..end2]
Let result[1..end2-start1+1] be a new array
indexL = start1;
indexR = start2;
indexRes = 1;
; // Go through both subarrays and pick the minimum element.
  Stop when one of the subarrays runs out.
while indexL ≤ end1 and indexR ≤ end2 do
|   if A[indexL] < A[indexR] then
|   |   result[indexRes] = A[indexL];
|   |   indexL = indexL + 1;
|   end
|   else
|   |   result[indexRes] = A[indexR];
|   |   indexR = indexR + 1;
|   end
|   indexRes = indexRes + 1;
end
; // Copy the remaining subarray over
while indexL ≤ end1 do
|   result[indexRes] = A[indexL];
|   indexL = indexL + 1;
|   indexRes = indexRes + 1;
end
while indexR ≤ end1 do
|   result[indexRes] = A[indexR];
|   indexR = indexR + 1;
|   indexRes = indexRes + 1;
end
; // Copy the result back
indexRes=1;
for i = start1 to end2 do
|   A[i] = result[indexRes];
end
```

3.5 Mergesort

Input: An array A
Input: integers start, end
if *end-start* ≤ 1 **then**
| **return***;*
end
mid = floor((start + end) /2) ;
mergesort(A, start, mid);
mergesort(A, mid+1, end);

4 Algorithm complexities

4.1 Running Time and Space Complexity

Definition: The running time of an algorithm A on a particular input is the number of primitive operations A performs when run on the input.

Definition: The space complexity of an algorithm A on a particular input is the amount of extra space the algorithm requires to run. Note: local variables count as $\Theta(1)$ space.

4.2 Best, Worst, Average Case

Definition: The best case time of an algorithm is the minimum number of primitive operations the algorithm requires over all possible inputs.

Definition: The worst case time of an algorithm is the maximum number of primitive operations the algorithm requires over all possible inputs.

Definition: The average case running time is the expectation of the running time over all possible inputs given a particular probability distribution.

4.3 $O, \Omega, \Theta, \omega, o$

Definition: We say that one function $f(n)$ is $O(g(n))$ if there exist constants $c \in \mathbb{R}^+, n_0 \in \mathbb{N}$ such that $f(n) \leq cg(n)$ for all $n \geq n_0$.

Example: $n \in O(n^2)$ why?

Definition: We say that one function $f(n)$ is $\Omega(g(n))$ if there exist constants $c \in \mathbb{R}^+, n_0 \in \mathbb{N}$ such that $f(n) \geq cg(n)$ for all $n \geq n_0$.

Definition: We say that one function $f(n)$ is $\Theta(g(n))$ if there exist constants $c_1, c_2 \in \mathbb{R}^+, n_0 \in \mathbb{N}$ such that $c_1 g(n) \leq f(n) \leq c_2 g(n)$ for all $n \geq n_0$.

limit definitions

Definition: We say that one function $f(n)$ is $O(g(n))$ if $\lim\limits_{n\to\infty} \dfrac{f(n)}{g(n)} = c$, for some $c \geq 0$

Definition: We say that one function $f(n)$ is $\Omega(g(n))$ if $\lim\limits_{n\to\infty} \dfrac{f(n)}{g(n)} = c$, for some $c > 0$ or $\lim_{n\to\infty} \frac{f(n)}{g(n)} \to \infty$.

Definition: We say that one function $f(n)$ is $\Theta(g(n))$ if $\lim_{n\to\infty} \frac{f(n)}{g(n)} = c$, for some $c > 0$

Talk about o and ω.

5 Theorems

1. $\Theta(f(n)) = \Omega(f(n)) \cap O(f(n))$
2. Θ is an equivalence relation.

6 Recurrences

1. substitution
2. recursion trees + substitution
3. Master theorem

7 Heaps

1. definition
2. inserting
3. removing
4. building a heap
5. heapsort