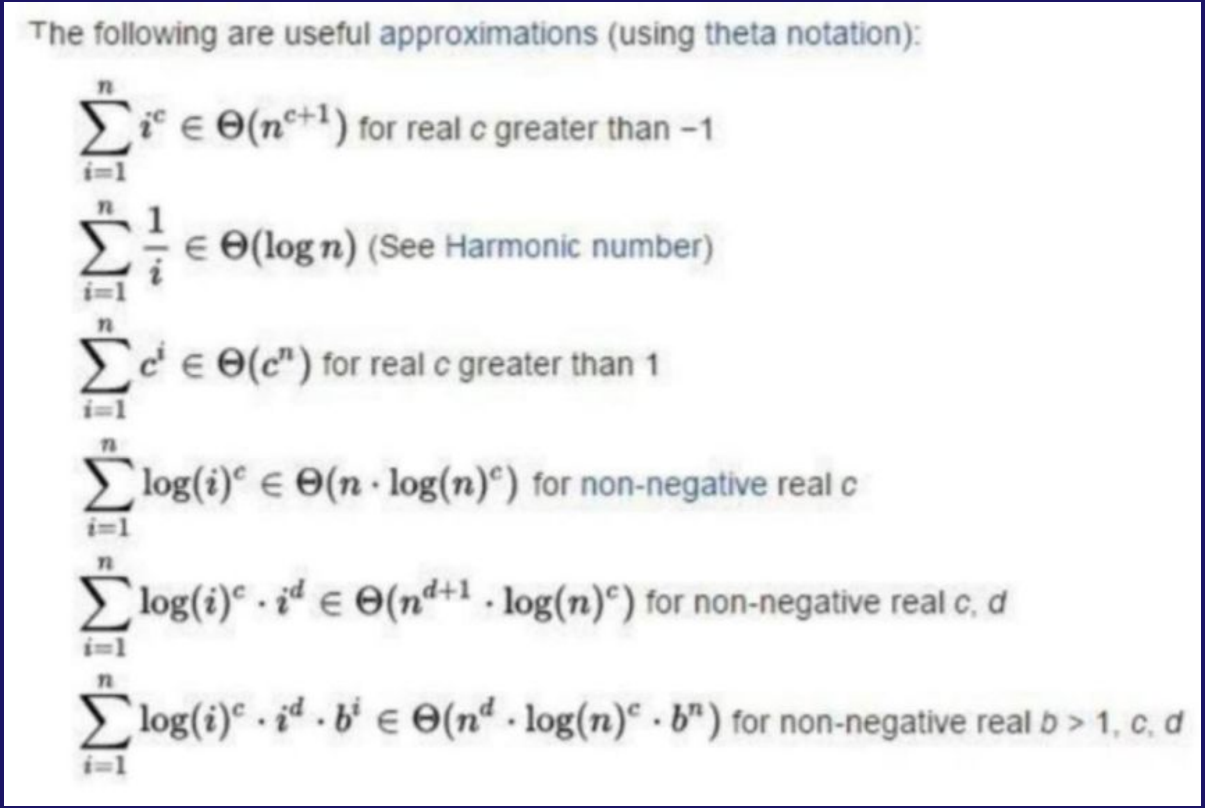
CS2040

Midterm Review

Nicholas **Russell** Saerang (russellsaerang@u.nus.edu)

Complexity



The following are useful approximations (using theta notation):

$$\sum_{i=1}^{n} i^c \in \Theta(n^{c+1})$$ for real c greater than -1

$$\sum_{i=1}^{n} \frac{1}{i} \in \Theta(\log n)$$ (See Harmonic number)

$$\sum_{i=1}^{n} c^i \in \Theta(c^n)$$ for real c greater than 1

$$\sum_{i=1}^{n} \log(i)^c \in \Theta(n \cdot \log(n)^c)$$ for non-negative real c

$$\sum_{i=1}^{n} \log(i)^c \cdot i^d \in \Theta(n^{d+1} \cdot \log(n)^c)$$ for non-negative real c, d

$$\sum_{i=1}^{n} \log(i)^c \cdot i^d \cdot b^i \in \Theta(n^d \cdot \log(n)^c \cdot b^n)$$ for non-negative real $b > 1$, c, d

Complexity

Master Theorem (simplified)

$$T(n) = aT\left(\frac{n}{b}\right) + O\left(n^k\right)\log^m n$$

$$= O\left(n^k\right) \text{ if } a < b^k$$

$$= O\left(n^k \log^{m+1} n\right) \text{ if } a = b^k$$

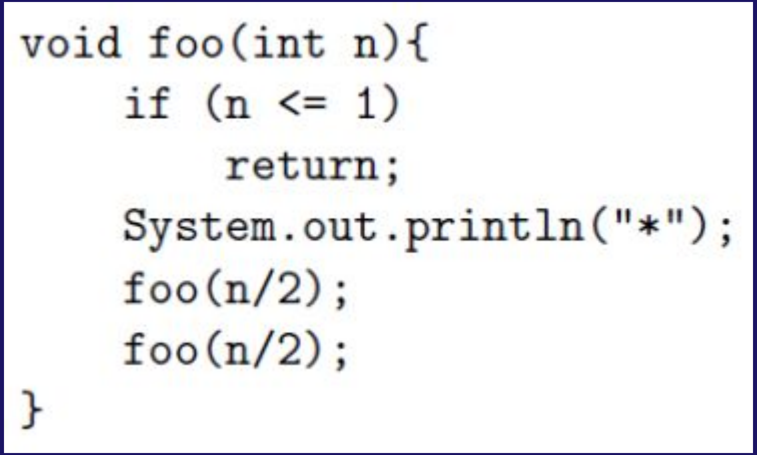
$$= O\left(n^{\log_b a}\right) \text{ if } a > b^k$$

Assumption: m >= 0

Example

a = b = 2, k = m = 0

So, the time complexity is O(n).



```
void foo(int n){
    if (n <= 1)
        return;
    System.out.println("*");
    foo(n/2);
    foo(n/2);
}
```

Complexity

Master Theorem (simplified)

$$T(n) = aT\left(\frac{n}{b}\right) + O\left(n^k \log^m n\right)$$
$$= O\left(n^k\right) \text{ if } a < b^k$$
$$= O\left(n^k \log^{m+1} n\right) \text{ if } a = b^k$$
$$= O\left(n^{\log_b a}\right) \text{ if } a > b^k$$

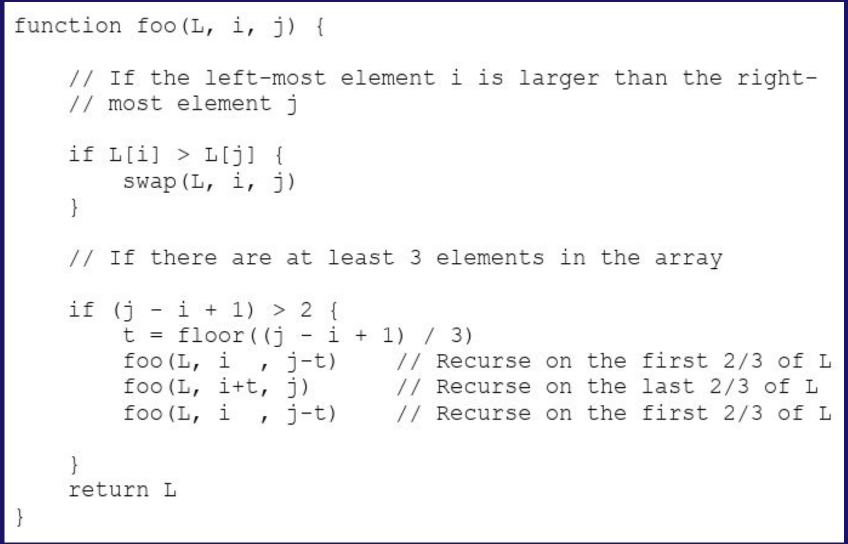
Assumption: m >= 0

Example (CS2040S 21/22 S1Q9)

T(n) = 3T(2n/3) + O(1)

a = 3, b = 3/2, k = 0, m = 0

Falls to the third case: $O(n^{\log(3)/\log(3/2)})$



```
function foo(L, i, j) {

    // If the left-most element i is larger than the right-
    // most element j

    if L[i] > L[j] {
        swap(L, i, j)
    }

    // If there are at least 3 elements in the array

    if (j - i + 1) > 2 {
        t = floor((j - i + 1) / 3)
        foo(L, i  , j-t)    // Recurse on the first 2/3 of L
        foo(L, i+t, j)      // Recurse on the last 2/3 of L
        foo(L, i  , j-t)    // Recurse on the first 2/3 of L

    }
    return L
}
```

Complexity

Master Theorem (simplified)

$$
\begin{aligned}
T(n) &= aT\left(\frac{n}{b}\right) + O\left(n^k\right) \log^m n \\
&= O\left(n^k\right) \text{ if } a < b^k \\
&= O\left(n^k \log^{m+1} n\right) \text{ if } a = b^k \\
&= O\left(n^{\log_b a}\right) \text{ if } a > b^k
\end{aligned}
$$

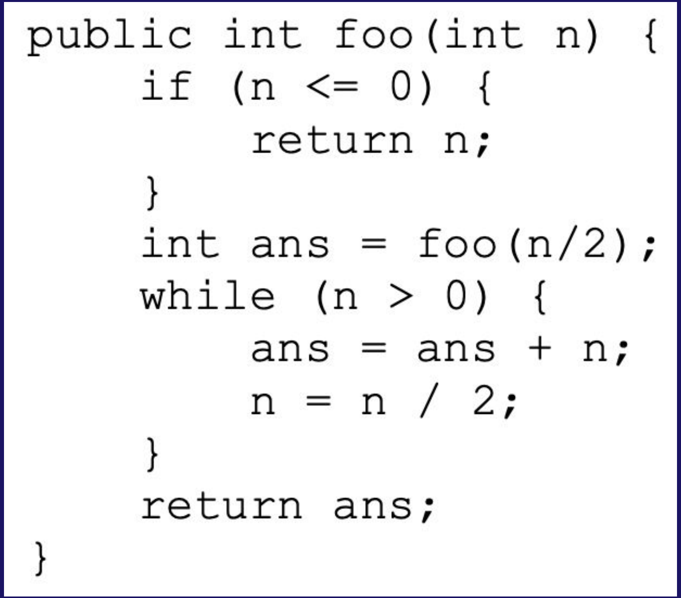
Assumption: m >= 0

Example (CS2040 20/21 S2Q1)

T(n) = T(n/2) + O(log n)

a = 1, b = 2, k = 0, m = 1

Falls to the second case: $O((\log n)^2)$



```
public int foo(int n) {
    if (n <= 0) {
        return n;
    }
    int ans = foo(n/2);
    while (n > 0) {
        ans = ans + n;
        n = n / 2;
    }
    return ans;
}
```

Sorting

1. Selection Sort: Every iteration, find largest item in the array, swap that item with the last element in the array
2. Bubble Sort: “Bubble” down the largest item to the end of the array in each iteration by examining the i-th and (i+1)-th item. Swap a[i] with a[i+1] if a[i] > a[i+1]
3. Insertion Sort: Start with first element in the array, pick the next element and insert into its proper sorted order. Repeat previous step for the rest of the elements in the array
4. Merge Sort: divide array into two, do MergeSort in the smaller arrays, merge the smaller arrays
5. Quick Sort: Choose pivot, divide array into 2 parts [<p][>=p], quick sort on both parts
6. Radix Sort: Treats each data to be sorted as a character string. In each iteration, organize the data into groups according to the next character in each data.

List, Stack, Queue

Tailed linked list

- maintain head and tail → insert, delete at head or tail can be done in O(1)
- could be used to implement deque/queue

Linked list, in general, could be used to implement stack

Expression Evaluation

Suppose we have
(5 + 7) * (12 / 4) - 5 = 31

Prefix: - * + 5 7 / 12 4 5
This uniquely identifies the expression even without the brackets.

Postfix: 5 7 + 12 4 / * 5 -
This also uniquely identifies the expression even without the brackets.

Uniqueness is not the case for infix expressions unless there are brackets!

Expression Evaluation

Prefix: - * + 5 7 / 12 4 5

Idea:
Use a stack and keep reading until the last two tokens are numbers

Simulation:
Read [- * + 5 7] to stack
Last two tokens are numbers, pop three times and evaluate
Stack is now [- * 12]
Continue reading [- * 12 / 12 4]
Last two tokens are numbers, pop three times and evaluate
Stack is now [- * 12 3]
Last two tokens are numbers, pop three times and evaluate
Stack is now [- 36]
Continue reading [- 36 5]
Last two tokens are numbers, pop three times and evaluate
Stack is now [31]

Expression Evaluation

Postfix: 5 7 + 12 4 / * 5 -

Idea:
Use a stack and keep reading until you see an operator

Simulation:
Read [5 7 +] to stack
Last token is an operator, pop three times and evaluate, stack is now [12]
Continue reading [12 12 4 /]
Last token is an operator, pop three times and evaluate, stack is now [12 3]
Continue reading [12 3 *]
Last token is an operator, pop three times and evaluate, stack is now [36]
Continue reading [36 5 -]
Last token is an operator, pop three times and evaluate, stack is now [31]

Expression Evaluation

Infix: (5 + 7) * (12 / 4) - 5

Idea:
Use the Shunting Yard Algorithm to give precedence value for each operator

Since Wikipedia has it… :)
https://en.wikipedia.org/wiki/Shunting_yard_algorithm

Probing

Keep jumping until you find an empty spot.
Always count from where you're currently at.

Linear probing:
+1, +1, +1, +1, …

Quadratic probing:
+1, +3, +5, +7, …

Modified linear probing:
+d, +d, +d, +d, … where d is some positive integer (> 0)

Double hashing: modified linear probing where $d = h_2(x)$

Probing

BUT if you choose to count from the starting point $h_1(x)$...
Some things have to change!

Linear probing:
+1, +2, +3, +4, …

Quadratic probing:
+1, +4, +9, +16, …

Modified linear probing:
+d, +2d, +3d, +4d, … where d is some positive integer (> 0)

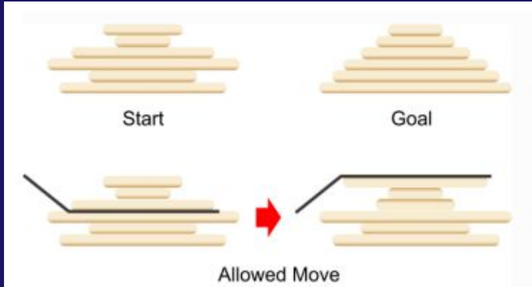
Double hashing: modified linear probing where $d = h_2(x)$

Famous Problems

#1 Pancake Flipping

Given stack of pancake with random order of diameter, find smallest number of flip to make the pancake stack ordered (smallest diameter on top)

Idea:
Flip the biggest pancake to the top then flip to bottom.
At the end of the i-th flip, i/2 largest pancake will be at the bottom.
T(n) = T(n-1) + 2, T(2) = 1 → Ans: 2n-3 flips



Famous Problems

#2 Two-sum

Find a pair of numbers from array A whose sum is equal to n.

Idea:
Add every item in A into hash table. For every number i in A, find n - i in the hash table, if it exists we find 2 elements in A that sums up to n (corner case, need to check if both elements are equal)

Time complexity is O(n) given O(1) time to hash.

Famous Problems

#3 Queue with 2 Stacks

CS2040 Midterm AY20/21 Sem 2 Q14 + Extra Practice 3

Let S1 and S2 be two stacks. Simulate the queue ADT.

- Offer:
  - Push item to S1, time is O(1)
- Poll:
  - If both stacks are empty, return null → O(1)
  - If S2 is empty, transfer S1 to S2 then pop S2 → amortized O(1)
  - Else, pop S2 and return that value → O(1)

Famous Problems

#4 Largest Area Under Histogram

Available at Extra Practice 3
Similar variants:

- CS2040 AY19/20 Sem 1 Q6b
- https://open.kattis.com/problems/stol (Solution by TA Eric)

Idea:

- Start with an empty stack
- For i = 0,…,n-1, do the following:
    - if stack is empty or hist[i] > top of stack, push i to stack
    - else, keep removing the top of the stack and update the maximum area every time with the area of the rectangle where the removed bar is the height and the index difference is the width

In the end, keep removing the top of the stack and update the area the same way (highlighted in yellow)
Overall time complexity is O(n).

Famous Problems

#5 Quicksort Pivots

Given a result of a single Quicksort array partition, find the number of elements that could have been the pivot.

Available at VA quiz & https://open.kattis.com/problems/pivot.

Idea:

- Find the running maximum from the left and the running minimum from the right. Store each corresponding values in two arrays RMAX and RMIN. Now loop through i = 0,...,n-1.
- If RMAX[i - 1] < A[i] < RMIN[i + 1] (ignore inequality of index is out of bounds), then that element is a possible pivot. Note that RMAX[k] = the maximum of A[0…k] and RMIN[k] = the minimum of A[k…(n-1)].

Overall time complexity is O(n) due to 3 array traversals.

Famous Problems

#6 Next Greater Element (NGE)

The next greater element for an array element x is the first greater element on the right side of x in the array. Find the next greater element of each element in the array.

Idea:

- Let S be a stack. Push the first element to S.
- For every remaining element in the array:
  - Mark the current element as *next*.
  - If S is not empty, compare top of S with *next*.
  - If *next* > top, pop top from S. *next* is the next greater element for top.
  - Keep popping from S while the popped element is smaller than *next*. *next* becomes the next greater element for all the popped elements.

Finally, push *next* in S. The remaining elements will have no NGE.
Overall time complexity is O(n).

Famous Problems

#7 Inversion Index (quite advanced)

Count the number of swaps needed by bubble sort to sort an array A.
See https://www.geeksforgeeks.org/counting-inversions/

Idea:

- Modify the merge sort algorithm
- Recursion on two halves on the array: numInversion(arr) = numInversion(left) + numInversion(right) + extra, what is this extra?
  - The inversion across both halves of the array!
- Notice during the merging process
  - Suppose i is the index for the first half, and j is an index of the second half. If A[i] is greater than A[j], then there are (mid - i) inversions because the left and right subarrays are sorted, so all the remaining elements in left-subarray (A[i+1], A[i+2], …, A[mid]) will be greater than A[j].

Overall time complexity is O(n log n).