University of Pittsburgh

Algorithms and Data Structures 1
CS 0445

Fall 2022

Sherif Khattab

ksm73@pitt.edu

(Slides are adapted from Dr. Ramirez's and Dr. Farnan's CS 0445 slides.)

Announcements

- Upcoming Deadlines:
  - Homework 7: this Friday @ 11:59 pm
  - Lab 6: next Monday 10/31 @ 11:59 pm
  - **Assignment 2: Monday 11/7 @ 11:59 pm**
- Live Support Session for Assignment 2
  - This Friday 10/28 @ 5:30 pm
- Lab Solutions for Lab 1-Lab 5 available in the handouts repository
- Live QA Session on Piazza every Friday 4:30-5:30 pm

Announcements

- Midterm grades
  - You can reattempt some of the questions
  - Use the “Request Regrade” feature in GradeScope
  - A maximum of 10 points
  - Deadline: Thursday 11/10 @ 11:59 pm
- Assignment 2 :
  - Running-time of both getFrequencyOf methods is O(frequency of (first) item)

Previous Lecture …

- Amortized running-time analysis
- More recursion examples
  - binary search
  - finding words in a grid of letters
- Running-time analysis of recursive Binary Search

Today …

- Muddiest Points

Muddiest Points

- Q: [In Binary Search analysis,] for simplicity we assume n = 2^k, but what if we have 14, which could not be presented as 2^k. How are we dealing with that?
- We use the next power of 2 (i.e., 16 in your example)
- This gives us an upper bound on the running-time
- Worst-case running-time of Binary search is O(ceiling($\log_2$(n)))

Muddiest Points

- **Q: In what situations is Amortized runtime useful**
- When the running-time of a method varies independently of the values of the input
  - e.g., add() in ResizableArrayBag:
    - O(1) if the array is not full; O(n) when the array is full
    - these cases are independent of the values of the items in the array

Muddiest Points

- Q: Why is the average asymptotic runtime equivalent to the worst-case asymptotic runtime?
- For linear search, both average-case and worst-case running-time were O(n)
- In general, They are not the same

Muddiest Points

- **Q: Recursion Trees**
- **Q: I am still a little confused about tracing exactly through a recursive method. Could you go over an example when you go line by line through a method?**
- **Q: For recursion, what happens if you have more than one recursive call in a method?**
- A recursion tree represents the recursive calls that occur during the execution of a recursive algorithm
- Each node in the recursion tree represents one call
- Total running-time =

  (total number of calls/nodes) * (time spent in each call)
- Can be built from the algorithm or from the recurrence relation of its running time
- Let's see an example on the board
  - Finding the nth Fibonacci number
  - Binary Search

Muddiest Points

- **Q: when is the best/worst times to use recursion?**
- Recursion is used when
  - iterative solution is complicated
    - e.g., Towers of Hanoi, Eight-Queens, and Word Search
  - the problem has smaller non-repeating subproblems of the same nature as the original problem
    - e.g., exponentiation (x^y)
  - recursive solution is more natural/intuitive
    - e.g., algorithms on recursive data structures (e.g., Trees (not covered))
- Recursion is carefully used when
  - the problem has smaller repeating subproblems
    - e.g., Finding the nth Fibonacci number

Muddiest Points

- **Q: I was unsure of how exactly the tower of Hanoi runtime translated to a big O notation of 2^n. Is it because every time you move up from the level one step, you go through two instances of the previous step again?**
- Many repeated subproblems!
- Let's draw on the board the recursion tree of the Towers of Hanoi to gain an insight

Muddiest Points

- **Q: I am just worried about implementing recursion and that I will not be able to visualize it in a way to make my code run successfully.**
- That's a valid and common concern
- There are multiple ways to visualize recursion
  - recursion tree
  - runtime stack
- However, thinking of the imaginary friend who can solve the same but smaller problem often helps

Muddiest Points

- Q: How do you implement backtracking as a part of your method?
- First, the problem has to have a certain structure:
  - a sequence of decisions to make
  - each decision has multiple options
  - when you choose an option for one decision, you are not sure if it is the best option
    - may need to try a different option after making further decisions
    - this happens regardless of the order you take the decisions
- Let's see the backtracking template again
- Be careful that for some of the problems with the above structure, the order of taking the decisions may avoid backtracking
  - You will see examples in CS 1501

Backtracking Framework

```
void solve(current decision, partial solution) {
  for each choice at the current decision {
    if choice is valid {
      apply choice to partial solution
      if partial solution a valid solution
        report partial solution as a final solution
      if more decisions possible
        solve(next decision, updated partial solution)
      undo changes to partial solution
    }
  }
}
```

Muddiest Points

- **Q: I am just wondering when assignment 2 will drop because we already are down to 2 weeks to complete it.**
- Although you have 2 weeks instead of 3 weeks in the tentative schedule on the syllabus, Assignment 2 has been reduced in difficulty/amount of work to match the reduced time

Muddiest Points

- **Q: I understand the recurrence relation but can you go over the iterative approach again?**
- I think you mean changing tail recursion to iteration
- General procedure
  - have the if statement around the recursive case
  - turn if into while
  - replace tail-recursive call by a sequence of parameter=argument assignment statements
- Let’s turn recursive Binary Search into iterative!

Muddiest Points

- **Q: proof by induction**
- Let’s prove the running time of recursive Binary Search using induction

Muddiest Points

- **Q: I don’t understand how the display backwards recursive method works. Is it even efficient if it has to go to the end of the linked list first and then all the way back to display every node?**
- Let’s trace it and analyze its running time