Review - Recursive algorithms

- To write a recursive algorithm:
  - Find how the problem can be broken up in one or more smaller problems of the same nature
  - Remember the base case!
- Usually, better running times are obtained when the size of the subproblems are approximately equal
  - $\text{power}(a,n) = a \times \text{power}(a,n)$ \(\Rightarrow\) \(O(n)\)
  - $\text{power}(a,n) = (\text{power}(a,n/2))^2 \Rightarrow O(\log n)$
- Fibonacci, BinarySearch, Power, Integer multiplication, MergeSort

Recursive algorithms

- Fibonacci
- BinarySearch
- Power
- Integer multiplication
- MergeSort

Merge Sort

Divide

4 3 2 1

4 3
4

2 1
2

1

3 4

Merge

1 2 3 4

Induction proofs

- To prove that a proposition $P(n)$ holds for all $n \geq a$:
  - Base case:
    Prove that $P(a)$ holds
  - Induction step on $n$:
    Induction Hypothesis: Assume $P(b)$ holds for all $b \leq n$
    Prove that I.H. implies that $P(n+1)$ holds

Generalized induction proofs

- To prove that a proposition $P(n)$ holds for all $n \geq a$:
  - Base case:
    Prove that $P(a), P(a+1), \ldots$ (as many as needed) hold
  - Induction step on $n$:
    Induction Hypothesis: Assume $P(b)$ holds for all $b \leq n$
    Prove that I.H. implies that $P(n+1)$ holds

Running time

- Primitive operations
  - Running time is constant, indep. of problem size
    - Assigning a value to a variable
    - Calling a method, returning from a method
    - Arithmetic operations, comparisons
    - Indexing into an array
    - Following object reference
  - Conditions
  - Running time = Number of primitive operations
  - Loops: Sum the running time of each iteration
    - $\text{findMin}, \text{selectionSort}, \text{insertionSort}$
Recurrences

- For recursive algorithms, we express the running time \( T(n) \) for an input of size \( n \) as a function of \( T(a) \) for some \( a < n \)
- Example:
  - Binary search: \( T(n) = T(n/2) + a \)
  - RecursiveFastMult.: \( T(n) = 3 \, T(n/2) + c \, n + d \)

Solving recurrences

- Solving recurrence \( \Rightarrow \) give explicit formula for \( T(n) \)
- Substitution method:
  - Replace occurrences of \( T() \) by their value
  - Repeat until pattern emerges
- Recursion tree:
  - Sum total overhead incurred at each recursion level
- Prove by induction that guess is correct

Big-Oh notation

\( g(n) \) is \( O(f(n)) \) iff there exist constants \( c \) and \( n_0 \) such that \( g(n) \leq c \, f(n) \) for all \( n \geq n_0 \)

- \( f(n) \) is \( \Omega(g(n)) \) iff \( g(n) \) is \( O(f(n)) \)
- \( f(n) \) is \( \Theta(g(n)) \) iff \( f(n) \) is \( O(g(n)) \) and \( f(n) \) is \( \Omega(g(n)) \)

Hierarchy of big-Oh classes

Big-Oh notation

Simplification rules

- If \( f_1(n) \in O(g(n)) \) and \( f_2(n) \in O(g(n)) \) then \( f_1(n) + f_2(n) \in O(g(n)) \)
- If \( f_1(n) \in O(g(n)) \) then \( k \cdot f_1(n) \in O(g(n)) \)
- If \( f_1(n) \in O(g(n)) \) and \( f_2(n) \in O(h(n)) \) then \( f_1(n) \cdot f_2(n) \in O(g(n) \cdot h(n)) \)

Test of the limit of the ratio

\[
L = \lim_{n \to \infty} \frac{a_{n+1}}{a_n}
\]

- If \( L < 1 \) then the series converges absolutely;
- If \( L > 1 \) then the series does not converge;
- If \( L = 1 \) or not existing, then the test is inconclusive.
Log identities

- \( \log(a \cdot b) = \log(a) + \log(b) \)
- \( \log(a^n) = n \log(a) \)
- \( \log_b(n) = \frac{\log_a(n)}{\log_a(b)} \)
- \( a^{\log_b(n)} = n^{\log_b(a)} \)

Running time of a For loop

```java
for (i=1; i<N; i=i*2) { ... }
```

Value of \( i \) after \( k \) iterations: \( 2^k \)

We have \( i < N \Rightarrow 2^k < N \Rightarrow k < \log_2(N) \).

There is less than \( \log_2(N) \) iterations, and the running time of this loop is \( O(\log(n)) \).

Master theorem

Recipe for recurrences of the form

\[
T(n) = a \cdot T\left(\frac{n}{b}\right) + f(n)
\]

- Size of problem
- Size of sub-problems
- Number of problems in recursion
- cost

Master theorem

- Recipe for recurrences of the form
  \( T(n) = a \cdot T(n/b) + f(n) \)
- Choose one of three cases depending on the comparison between \( f(n) \) to \( n^{\log_b(a)} \)
- Usually gives big-Theta notation for \( T(n) \)

Linked lists

- Implementation (node: value, next)
- Basic operations
  - getFirst(), get(n), getLast()
  - removeFirst(), removeLast(), remove(o)
  - addFirst(o), addLast(o), add(o)
  - empty(), size()
- Advantages and disadvantages over arrays
Stacks

- Last-in First-out data structure
- Operations: push(o), pop(), top()
- Applications:
  - Back button on browser
  - Checking parentheses
  - Execution of a program in the JVM

Deques with rotating arrays

Operations on deques with Array

- Queue() {
  L = new Array[N];
  head = tail = -1;
}
- isEmpty() {
  if (head == -1) && (tail == -1) return true;
  else return false;
}
- isFull() {
  if ((head - tail) % N == 1) return true;
  else return false;
}
- Enqueue(o) throw Exception {
  if (isFull()) { throw new Exception("Full stack") }
  if (isEmpty()) { head = tail = 0; }
  else {
    tail = ( ( tail + 1 ) % N );
  }
  L[tail] = o;
}
- Dequeue() {
  if (isEmpty()) { throw new Exception("Empty stack") }
  Object o = L[head];
  if (((head - tail) % N) == 1) { head = tail = -1; }
  else { head = ( ( head + 1 ) % N ); }
  return o;
}

Exam instructions

- Open book, open notes, calculators ok.
- Arrive early. The exam starts at 6pm sharp!
- Sign the "sign-up sheet".
- Don't get stuck on one problem. First answer the questions you feel more confident with