CS202 - Algorithm Analysis
Data Structure & Algorithm 2

Aravind Mohan

Allegheny College

March 24, 2021

A Follow-up on Practicals 3

4	5	7	3	2	1	6	8

```
i = 0 & h = −1
done = false
span = [1,0,0,0,0,0,0,0]
stack = [0]
```

```
i = 1 & h = −1
done = false
span = [1,2,0,0,0,0,0,0]
stack = [1]
```

```
i = 2 & h = −1
done = false
span = [1,2,3,0,0,0,0,0]
stack = [2]
```

```
i = 3 & h = 2
done = true
span = [1,2,3,1,0,0,0,0]
stack = [2,3]
```

Problem-3

A Follow-up on Practicals 3

4	5	7	3	2	1	6	8

```
i = 4 & h = 3
done = true
span = [1,2,3,1,1,0,0,0]
stack = [2,3,4]
```

```
i = 5 & h = 4
done = true
span = [1,2,3,1,1,1,0,0]
stack = [2,3,4,5]
```

```
i = 6 & h = 2
done = true
span = [1,2,3,1,1,1,4,0]
stack = [2,6]
```

```
i = 7; h = -1;
done = false
span = [1,2,3,1,1,1,4,8];
stack = [7]
```

Problem-3

Discussion Based On ...

Sedgewick 1.3, Queues, 2.1 Insertion Sort

Queue

A line of people standing in a ticket counter is similar to a **Queue**.

What is a Queue ADT?

- A queue differs from stack in that its insertion and removal routines follows first in first out (FIFO) principal.
- Elements can be inserted at any time, but only the element which has been in the queue longest can be removed.
- Elements are inserted in the rear (enqueued) and removed from the front (dequeued).

Queue ADT Operations

- A Queue is an Abstract Data Type that supports four main methods:
  - **new():**ADT - Creates a new queue.
  - **enqueue(Q:ADT, o:element):**ADT - Inserts object o at the rear of the queue Q.
  - **dequeue(Q:ADT):**ADT - Removes the object from the front of the queue; if the queue is empty an error occurs.
  - **front(Q:ADT):**element - returns, but does not remove ,the front element; an error occurs if the queue is empty.

Queue ADT Supporting Operations

- **size(Q:ADT):**integer - Returns the number of objects in queue Q.
- **isEmpty(Q:ADT):**boolean - Indicates if queue Q is empty.

Axioms on Queue

An axiom, is a statement that is taken to be true, to serve as a premise or starting point for further reasoning and arguments.

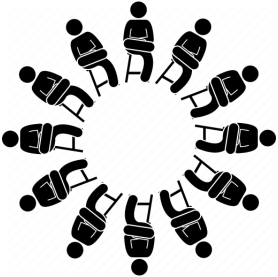
The following axioms dictates the scope of the operations in the queue.

- **Front(Enqueue(new(),v)) = v**
- **Dequeue(Enqueue(new(),v)) = new()**
- **Front(Enqueue(Enqueue(Q,u),v)) = Front(Enqueue(Q,u))**
- **Dequeue(Enqueue(Enqueue(Q,u),v)) = Enqueue(Dequeue(Enqueue(Q,u)),v)**

Algorithmic Problem

Ready for another Algorithmic Problem?

Josephus Problem

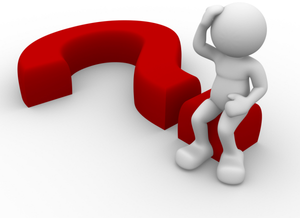


Problem Definition: A group of n people are standing in a circle, numbered consecutively clockwise from 1 to n. Starting with person no. 2, we remove every other person, proceeding clockwise. For example, if n = 6, the people are removed in the order 2, 4, 6, 3, 1, and the last person remaining is no. 5. Let $j(n)$ denote the last person remaining. Find some simple way to compute $j(n)$ for any positive integer $n > 1$.

Josephus Problem

Queue is an apt data structure to solve this problem.

So how to write an algorithm to solve this problem?



Please dont see next slide. **Think** yourself first!

Josephus Problem Algorithm

Algorithm - Josephus(Q)

Input: an n-element queue Q of values such that Q[i] is the value connected to a person at position i in the circle.

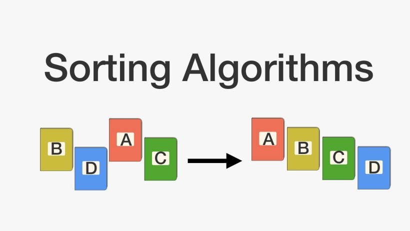
Output: the value connected to a person who is the last one remaining in the circle.

```
1: while Q.size() > 1) do
2:     Q.enqueue(Q.dequeue)
3:     Q.dequeue()
4: end while
5: return Q.front()
```

Thinking Exercise

- What is the worst-case asymptotic running time of this algorithm?
- How to transform this algorithm into its code/program equivalent?

Sorting



List Data Structure

- Definition: Organizing a set of data items into either ascending or descending order.
  - Internal sorting - main memory
  - External sorting - secondary storage

Applications of Sorting

(a)

(b)

(c)

How to measure the efficiency?

- Asymptotic Analysis
- Counting the number of key comparisons and the number of moves

Sorting More Formally

Input: Sequence of numbers $a_1, a_2, a_3, \cdots, a_n$

Output: A permuatation of the input sequence, $b_1, b_2, b_3, \cdots, b_n$

Sorting More Formally

Correctness (requirements for the output)

For any given input the algorithm halts with the output

- $b_1 < b_2 < b_3 < \cdots < b_n$
- $b_1, b_2, b_3, \cdots, b_n$ is a permutation of $a_1, a_2, a_3, \cdots, a_n$

Factors that affect efficiency?

- Number of data items (N)
- How (partially) sorted they are?
- Quality of the algorithm

Insertion Sort Algorithm

Strategy:

- Start empty handed
- Insert a card in the right position of the already sorted hand
- Continue until all cards are inserted or sorted

Insertion Sort Algorithm

Algorithm - Insertion(A)

Input: an n-element un-sorted array A of integer values.

Output: an n-element sorted array A of integer values.

for $i = 1$ to n **do**
  $key \leftarrow A[i]$
  $j \leftarrow i - 1$
  while $j >= 0$ and $A[j] > key$) **do**
    $A[j+1] \leftarrow A[j]$
    $j \leftarrow j - 1$
  end while
  $A[j+1] \leftarrow key$
end for

Insertion Sort Example

Input: [5,4,3,2,1])

```
Phase:1
[5, 5, 3, 2, 1]
[4, 5, 3, 2, 1]
------------------------------
Phase:2
[4, 5, 5, 2, 1]
[4, 4, 5, 2, 1]
[3, 4, 5, 2, 1]
------------------------------
Phase:3
[3, 4, 5, 5, 1]
[3, 4, 4, 5, 1]
[3, 3, 4, 5, 1]
[2, 3, 4, 5, 1]
------------------------------
Phase:4
[2, 3, 4, 5, 5]
[2, 3, 4, 4, 5]
[2, 3, 3, 4, 5]
[2, 2, 3, 4, 5]
[1, 2, 3, 4, 5]
------------------------------
[1, 2, 3, 4, 5]
```

Insertion Sort Example

Input: [1,2,3,4,5])

```
Phase:1
[1, 2, 3, 4, 5]
-----------------------------
Phase:2
[1, 2, 3, 4, 5]
-----------------------------
Phase:3
[1, 2, 3, 4, 5]
-----------------------------
Phase:4
[1, 2, 3, 4, 5]
-----------------------------
[1, 2, 3, 4, 5]
```

- L_2, L_3, and L_8 executes one time each for every iteration.

$$1 + 1 + 1 + 1 + \cdots + n - 1 = \textbf{n-1}$$

- L_5 and L_6 executes in the sequence shown below for all iterations combined.

$$1 + 2 + 3 + 4 + \cdots + n - 1$$
$$= \frac{n \times (n-1)}{2} = \frac{n^2}{2} = \mathbf{n^2}$$

- Total execution time $= (n-1) + n^2 = \mathbf{O(n^2)}$

Worst Case Analysis

UPDATE: Variation to previous slide, n is total number of elements.

- L_2, L_3, and L_8 executes one time each for every iteration.

$$1 + 1 + 1 + 1 + \cdots + n = \mathbf{n}$$

- L_5 and L_6 executes in the sequence shown below for all iterations combined.

$$1 + 2 + 3 + 4 + \cdots + n$$
$$= \frac{n \times (n+1)}{2} = \frac{n^2}{2} = \mathbf{n^2}$$

- Total execution time $= (n) + n^2 = \mathbf{O(n^2)}$

Asymptotically both are similar.

Best Case Analysis

- L_2, L_3, and L_8 executes one time each for every iteration.

$$1 + 1 + 1 + 1 + \cdots + n - 1 = \textbf{n-1}$$

- L_5 and L_6 executes **0** times in total for all iterations combined.
- Total execution time $= 0 + (n - 1) =$ **O(n)**

Average Case Analysis

- L_2, L_3, and L_8 executes one time each for every iteration.

$$1 + 1 + 1 + 1 + \cdots + n - 1 = \mathbf{n\text{-}1}$$

- L_5 and L_6 executes in the sequence shown below for all iterations combined.

$$\frac{1 + 2 + 3 + 4 + \cdots + n - 1}{2}$$

$$= \frac{n \times (n-1)}{4} = \frac{n^2}{4} = \mathbf{n^2}$$

- Total execution time $= (n-1) + n^2 = \mathbf{O(n^2)}$

Reading Assignment

Sedgewick 1.3 Queues, 2.1 Insertion Sort

Questions?

Please ask if there are any Questions!