CSE 250: Induction

Lecture 12

Sept 25, 2023

Reminders

- WA2 due Sun, Oct 1 at 11:59 PM
  - Released Monday
- Midterm 1 on Oct 2 in-class
  - Covers: Asymptotics, Sequences/Lists, Arrays, Linked Lists, Recursion
  - Bounds: Tight Upper/Lower, Unqualified vs Amortized

Sorted Lists

Sorting an Array

- **Bubble Sort**: $O(N^2)$
- **Merge Sort**: $O(N \log(N)))$ (probably?)

Induction

1. Generate a hypothesis.
2. Prove the hypothesis for the base case.
3. Prove the hypothesis inductively.
   - Assume that the hypothesis is true for a small case (e.g., $N-1$).
   - Prove that the hypothesis is true for a bigger case (e.g., N).

Induction

We showed there exists a c such that...

- $T_f(1) \leq 1 \cdot c$ (Base Case)
- if $T_f(N-1) \leq (N-1) \cdot c$ then $T_f(N) \leq N \cdot c$ (Inductive Proof)

So...

1. $T_f(1) \leq 1 \cdot c$ (Base Case)
2. $T_f(2) \leq 2 \cdot c$ (#1 + Inductive Proof)
3. $T_f(3) \leq 3 \cdot c$ (#2 + Inductive Proof)
4. $T_f(4) \leq 4 \cdot c$ (#3 + Inductive Proof)
5. $T_f(5) \leq 5 \cdot c$ (#4 + Inductive Proof)
6. $T_f(6) \leq 6 \cdot c$ (#5 + Inductive Proof)
7. ...

The proof holds for any $N \geq 1 \rightarrow T(N) \in O(N)$ ✓

Factorial

What is the complexity of Factorial?

```
1    public long factorial(long N)
2    {
3      if(N <= 1){ return 1; }
4      else { return N * factorial(N-1); }
5    }
```

Answer: $O(N)$[1]

How much memory does it use?

[1] Technically it's $\theta(N)$, but we haven't proven $T(N) \in \Omega(N)$

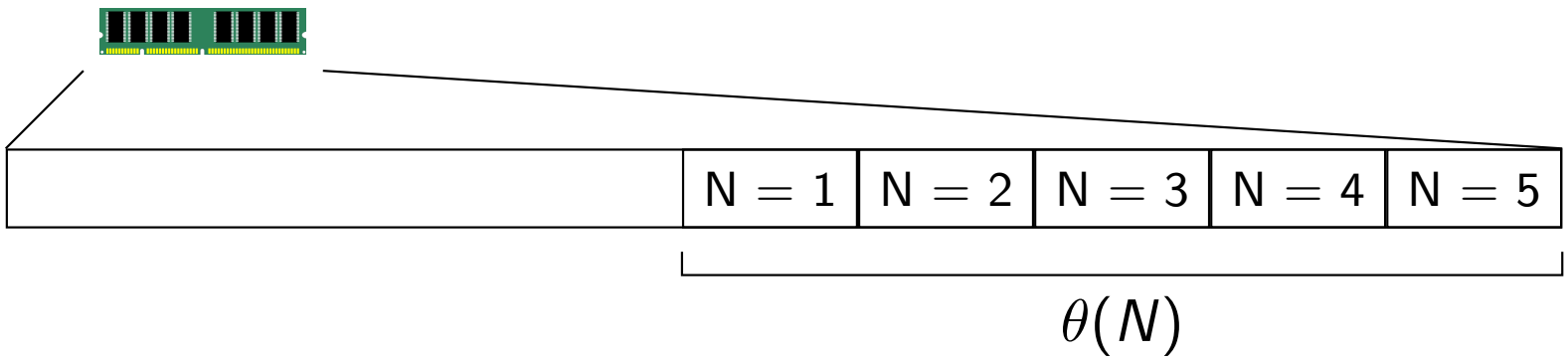
Stack Frames

Every time you call a function, it allocates some memory for local variables (e.g., `N`).

This chunk of memory is called a **Stack Frame**.

This is where the term `StackOverflowError` comes from.

Stack Frames



Factorial (as a loop)

```
1    public long factorial(long N)
2    {
3      long total = 1;
4      for(long i = N; i > 0; i--)
5      {
6        total *= i
7      }
8      return total
9    }
```

Factorial

Why does this work?

Factorial (as a loop)

```
1     public long factorial(long N)
2     {
3       if(N <= 1){ return 1; }
4       else { return N * factorial(N-1); }
5     }
```

Each call to factorial only makes **one** recursive call.

Factorial (as a loop)

- Is $\texttt{N} > 1$?
- Compute $\texttt{arg} = \texttt{N} - 1$
- Call `factorial(arg)`
- Compute $\texttt{N} \times$ result
- Return

← Requires stack frame

← Requires stack frame

← Requires stack frame

Factorial (as a loop)

```
1     public long factorial(long N, long total)
2     {
3       if(N <= 1){ return total; }
4       else { return factorial(N-1, N * total); }
5     }
```

Factorial (as a loop)

- Is $\texttt{N} > 1$?
- Compute $\texttt{arg1} = \texttt{N} - 1$
- Compute $\texttt{arg2} = \texttt{N} \times$ total
- Call `factorial(arg1, arg2)`
- Return

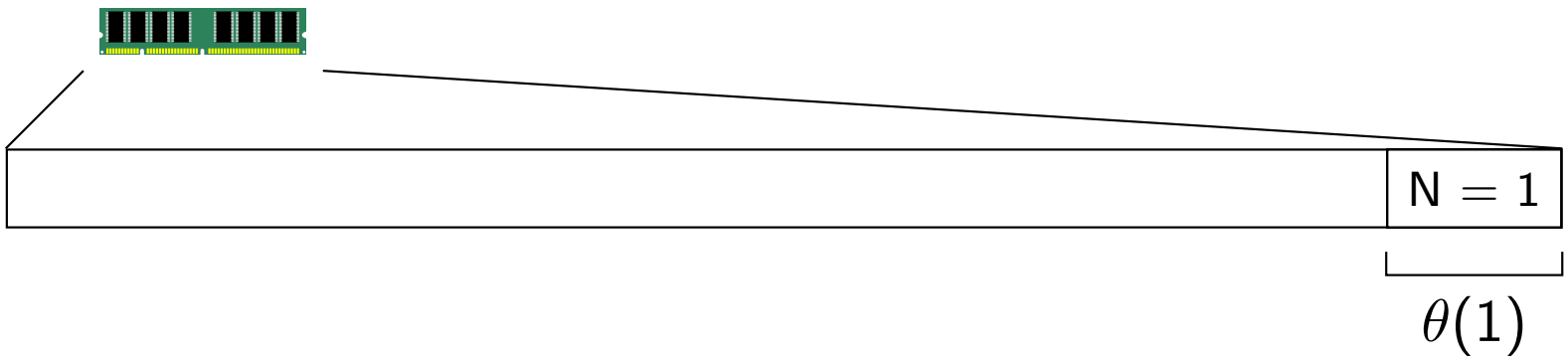
← Requires stack frame

← Requires stack frame

← Requires stack frame

← Stack frame unnecessary

Stack Frames



Tail Recursion

If the recursive call is the *last* operation before the return, most languages optimize the recursion away[2].

This is called **Tail Recursion**

[2] ... but not Java

Merge Sort

```
1   public ArrayList<E> mergeSort<E>(ArrayList<E> list)
2   {
3     if(list.size() > 2){
4       int splitIndex = input.size()/2;
5       ArrayList<E> left =
6         mergeSort(list.subList(0, splitIndex));
7       ArrayList<E> right =
8         mergeSort(list.subList(splitIndex, list.size));
9       return merge(left, right);
10    } else {
11      if((list.size() == 2) && (list.get(0) > list.get(1))){
12        E tmp = list.get(0);
13        list.set(0, list.get(1));
14        list.set(1, tmp);
15      }
16      return list;
17    }
18  }
```

Merge Sort

- If $N > 2$
  - Split — $O(1)$
  - $2\times$ `mergeSort(`$\frac{N}{2}$`)` — $2 \cdot T(\frac{N}{2})$
  - `merge(`N`)` — $O(N)$
- Otherwise
  - Sort 2 elements — $O(1)$

$$T_{merge}(N) = \begin{cases} O(N) + 2 \cdot T_{merge}(\frac{N}{2}) & \textbf{if } N > 2 \\ O(1) & \textbf{otherwise} \end{cases}$$

Merge Sort

$$T_{merge}(N) = \begin{cases} O(N) + 2 \cdot T_{merge}(\frac{N}{2}) & \textbf{if } N > 2 \\ O(1) & \textbf{otherwise} \end{cases}$$

Induction

- **Hypothesis:** $T_{merge}(N) \in O(N \cdot \log(N))$
  - $T_{merge}(N) \leq c \cdot N \log(N)$
- **Base Step:**
  - Show $T_{merge}(2) \leq c \cdot 2 \cdot \log(2) = c \cdot 2$
- **Inductive Step:**
  - Assume $T_{merge}(\frac{N}{2}) \leq c \cdot \frac{N}{2} \cdot \log\left(\frac{N}{2}\right)$
  - Show $T_{merge}(N) \leq c \cdot N \cdot \log(N)$

Base Step

$$T_{merge}(N) = \begin{cases} O(N) + 2 \cdot T_{merge}(\frac{N}{2}) & \textbf{if } N > 2 \\ O(1) & \textbf{otherwise} \end{cases}$$

$$T_{merge}(2) \overset{?}{\leq} c \cdot 2$$

$$O(1) \leq 2c$$

Inductive Step

$$T_{merge}(N) = \begin{cases} O(N) + 2 \cdot T_{merge}(\frac{N}{2}) & \textbf{if } N > 2 \\ O(1) & \textbf{otherwise} \end{cases}$$

Assume: $T_{merge}(\frac{N}{2}) \leq c \cdot \frac{N}{2} \cdot \log\left(\frac{N}{2}\right)$

$$T_{merge}(N) \stackrel{?}{\leq} c \cdot N \log(N)$$

$$O(N) + 2 \cdot T_{merge}\left(\frac{N}{2}\right) \stackrel{?}{\leq} c \cdot N \log(N)$$

$$O(N) + 2 \cdot c \cdot \frac{N}{2} \cdot \log\left(\frac{N}{2}\right) \stackrel{?}{\leq} c \cdot N \log(N)$$

Inductive Step

$$O(N) + 2 \cdot c \cdot \frac{N}{2} \cdot \log\left(\frac{N}{2}\right) \stackrel{?}{\leq} \cdot N \log(N)$$

$$O(N) + c \cdot N \cdot \log\left(\frac{N}{2}\right) \stackrel{?}{\leq} c \cdot N \log(N)$$

$$O(N) + c \cdot N \cdot (\log(N) - \log(2)) \stackrel{?}{\leq} c \cdot N \log(N)$$

$$O(N) + c \cdot N \cdot \log(N) \stackrel{?}{\leq} c \cdot N \log(N) + c \cdot N$$

$$O(N) \leq c \cdot N$$

Merge Sort

So... $T_{merge}(N) \in O(N \cdot \log(N))$

Quicksort

Merge Sort

- Split up the problem $(O(1))$
- Merge the sub-solutions $(O(N))$

Quicksort

- Split up the problem $(O(N))$
- Merge the sub-solutions $(O(1))$

Quicksort

- If the list has only 1 element, it's sorted. Return. $O(1)$
- Find the **median** value ('pivot'). $T_{pivot}(N) O(N \log(N))$
- Create a list of elements $<$ **medium**. $O(N)$
- Create a list of elements $>$ **medium**. $O(N)$
- Sort each of the two lists $2 \cdot T(\frac{N}{2})$
- Concatenate the lists $O(N)$ or $O(1)$

$$T_{merge}(N) = \begin{cases} 1 & \textbf{if } N \leq 1 \\ O(N) + 2\,T_{merge}(\frac{N}{2}) + T_{pivot}(N) N \log(N) & \textbf{otherwise} \end{cases}$$

Quicksort

Idea: If we pick a pivot value at random,
in expectation, half of the values will be lower.

The Worst-Case Pivot

What is the worst case runtime?

The Worst-Case Pivot

What if we always pick the worst pivot?

1. `[ 8, 7, 6, 5, 4, 3, 2, 1 ]`
2. `[ 7, 6, 5, 4, 3, 2, 1 ], 8, []`
3. `[ 6, 5, 4, 3, 2, 1 ], 7, [], 8, []`
4. `[ 5, 4, 3, 2, 1 ], 6, [], 7, [], 8, []`

...

- For each level, $O(N)$*work*
- At worst, $O(N)$ levels

Total: $T_{quicksort}(N) \in O(N^2)$

Quicksort

Expected Runtime

Is the worst case runtime representative?

No! (in typical cases, it will be faster)

Is there something we <u>can</u> say about the runtime?

Quicksort

If we pick the X'th largest element as a pivot, what is $T_{qs}(N)$?

- $X = 1$: $\theta(N) + T_{qs}(0) + T_{qs}(N-1)$
- $X = 2$: $\theta(N) + T_{qs}(1) + T_{qs}(N-2)$
- $X = 3$: $\theta(N) + T_{qs}(2) + T_{qs}(N-3)$
- $X = 4$: $\theta(N) + T_{qs}(3) + T_{qs}(N-4)$
- ...
- $X = N-1$: $\theta(N) + T_{qs}(N-2) + T_{qs}(1)$
- $X = N$: $\theta(N) + T_{qs}(N-1) + T_{qs}(0)$

$$T_{qs}(N) = \theta(N) + T_{qs}(X-1) + T_{qs}(N-X) \qquad (\text{for } X \in [1, N])$$

Probabilities

What is the chance we (randomly) pick $X = 1$? $P[X = 1] = \frac{1}{N}$

What is the chance we pick $X = \lfloor \frac{N}{2} \rfloor$? $P\left[X = \lfloor \frac{N}{2} \rfloor\right] = \frac{1}{N}$

What is the chance we pick $X = N$? $P[X = N] = \frac{1}{N}$

Probability

If I roll a d6 (a 6-sided die ) k times,
what is the average over all possible outcomes?

k=1

If I roll a d6 (a 6-sided die 🎲) 1 time...

Roll	Probability	Contribution
1	$\frac{1}{6}$	1
2	$\frac{1}{6}$	2
3	$\frac{1}{6}$	3
4	$\frac{1}{6}$	4
5	$\frac{1}{6}$	5
6	$\frac{1}{6}$	6

Expectation

$$\frac{1}{6} \cdot 1 + \frac{1}{6} \cdot 2 + \frac{1}{6} \cdot 3 + \frac{1}{6} \cdot 4 + \frac{1}{6} \cdot 5 + \frac{1}{6} \cdot 6 = 3.5$$

$$= \sum_i \textbf{Probability}_i \cdot \textbf{Contribution}_i$$

If X is a variable representing the outcome of the roll, we call this number the **expectation** of X, or $E[X]$.

$$E[X] = \sum_i P_i \cdot X_i$$

k = 2

If I roll a d6 (a 6-sided die 🎲) 2 times...
Does one roll affect the outcome of the other?

No: Each roll is a *independent* event.

Independent Events

If X and Y are random variables representing the outcome of each roll (i.e., independent random variables):

$$E[X + Y] = E[X] + E[Y]$$

$$= 3.5 + 3.6 = 7$$

Back to Quicksort

$$T_{qs}(N) = \begin{cases} \theta(1) & \textbf{if } N \leq 1 \\ T_{qs}(X-1) + T_{qs}(N-X) + \theta(N) & \textbf{otherwise} \end{cases}$$

... but X is random!

Expected Runtimes

$$E[T_{qs}(N)] = \begin{cases} \theta(1) & \textbf{if } N \leq 1 \\ E[T_{qs}(X-1) + T_{qs}(N-X) + \theta(N)] & \textbf{otherwise} \end{cases}$$

Expected Runtimes

$$E[T_{qs}(N)] = \begin{cases} \theta(1) & \textbf{if } N \leq 1 \\ E[T_{qs}(X-1)] + E[T_{qs}(N-X)] + E[\theta(N)] & \textbf{otherwise} \end{cases}$$

Expected Runtimes

$$E[T(X-1)]$$

$$= \sum_{i=1}^{N} P_i \cdot T(X_i - 1)$$

$$= \sum_{i=1}^{N} \frac{1}{N} \cdot T(i-1) \qquad (T(0) \text{ up to } T(n-1))$$

$$= \sum_{i=1}^{N} \frac{1}{N} \cdot T(n-i) \qquad (T(n-1) \text{ down to } T(0))$$

$$= E[T(N-X)]$$

Expected Runtimes

$$E[T_{qs}(N)] = \begin{cases} \theta(1) & \textbf{if } N \leq 1 \\ 2 \cdot E[T_{qs}(X-1)] + E[T_{qs}(N-X)] + \theta(N) & \textbf{otherwise} \end{cases}$$

The X we pick at each step is independent.

Expected Runtimes

$$E[T_{qs}(N)] = \begin{cases} \theta(1) & \textbf{if } N \leq 1 \\ 2 \cdot \left(\sum_{i=10}^{N-1} \frac{1}{N} E[T_{qs}(i-1)] \right) + \theta(N) & \textbf{otherwise} \end{cases}$$

Back to Induction! **Inductive Hypothesis:**
$E[T_{qs}(N)] \in O(N \log(N))$

$E[T_{qs}(N)] \leq c \cdot N \cdot \log(N)$

Quicksort's Runtime

$$E[T_{qs}(N)] = \begin{cases} c_1 & \textbf{if } N \leq 1 \\ 2 \cdot \left(\sum_{i=0}^{N-1} \frac{1}{N} E[T_{qs}(i)] \right) + c_2 N & \textbf{otherwise} \end{cases}$$

Base Case

$$E[T_{qs}(1)] \stackrel{?}{\leq} c \cdot 1 \cdot \log(1)$$

$$E[T_{qs}(1)] \stackrel{?}{\leq} c \cdot 1 \cdot 0$$

$$E[T_{qs}(1)] \not\leq 0 \qquad \ldots \text{ oops}$$

Quicksort's Expected Runtime

$$E[T_{qs}(N)] = \begin{cases} c_1 & \textbf{if } N \leq 1 \\ 2 \cdot \left(\sum_{i=0}^{N-1} \frac{1}{N} E[T_{qs}(i)] \right) + c_2 N & \textbf{otherwise} \end{cases}$$

Base Case (with $N_0 = 2$)

$$E[T_{qs}(2)] \stackrel{?}{\leq} c \cdot 2 \cdot \log(2)$$

$$2 \cdot \left(\sum_{i=0}^{1} \frac{1}{2} E[T_{qs}(i)] \right) + 2c_2 \stackrel{?}{\leq} c \cdot 2 \cdot \log(2)$$

$$\left(\sum_{i=0}^{1} \frac{1}{2} E[T_{qs}(i)] \right) + c_2 \stackrel{?}{\leq} c$$

Quicksort's Expected Runtime

$$\left(\sum_{i=0}^{1} \frac{1}{2} E[T_{qs}(i)]\right) + c_2 \overset{?}{\leq} c$$

$$\frac{1}{2}(c_1 + c_1) + c_2 \overset{?}{\leq} c$$

$$c_1 + c_2 \leq c$$

(true if we set $c \geq c_1 + c_2$)

Quicksort's Expected Runtime

$$E[T_{qs}(N)] = \begin{cases} c_1 & \textbf{if } N \leq 1 \\ 2 \cdot \left(\sum_{i=0}^{N-1} \frac{1}{N} E[T_{qs}(i)] \right) + c_2 N & \textbf{otherwise} \end{cases}$$

Inductive Step

Assume: $E[T_{qs}(N')] \leq c \cdot N' \log(N')$ for all $2 \leq N' \leq N$

Show: $E[T_{qs}(N)] \leq c \cdot N \log(N)$

$$2 \cdot \left(\sum_{i=0}^{N-1} \frac{1}{N} E[T_{qs}(i)] \right) + c_2 \stackrel{?}{\leq} c \cdot N \log(N)$$

$$2 \cdot \frac{1}{N} \left(\sum_{i=0}^{N-1} E[T_{qs}(i)] \right) + c_2 \stackrel{?}{\leq} c \cdot N \log(N)$$

Quicksort's Expected Runtime

$$2 \cdot \frac{1}{N} \left(\sum_{i=0}^{N-1} E[T_{qs}(i)] \right) + c_2 \stackrel{?}{\leq} c \cdot N \log(N)$$

$$2 \cdot \frac{1}{N} \left(E[T_{qs}(0)] + E[T_{qs}(1)] + \sum_{i=2}^{N-1} E[T_{qs}(i)] \right) + c_2 \stackrel{?}{\leq} c \cdot N \log(N)$$

$$2 \cdot \frac{1}{N} \left(2 \cdot c_1 + \sum_{i=2}^{N-1} E[T_{qs}(i)] \right) + c_2 \stackrel{?}{\leq} c \cdot N \log(N)$$

$$\frac{2}{N} \left(\sum_{i=2}^{N-1} E[T_{qs}(i)] \right) + \frac{2 \cdot c_1}{N} + c_2 \stackrel{?}{\leq} c \cdot N \log(N)$$

Quicksort's Expected Runtime

$$\frac{2}{N}\left(\sum_{i=2}^{N-1} E[T_{qs}(i)]\right) + \frac{2 \cdot c_1}{N} + c_2 \stackrel{?}{\leq} c \cdot N \log(N)$$

$$\frac{2}{N}\left(\sum_{i=2}^{N-1} c \cdot i \log(i)\right) + \frac{2 \cdot c_1}{N} + c_2 \stackrel{?}{\leq} c \cdot N \log(N)$$

$$\frac{2c}{N}\left(\sum_{i=2}^{N-1} i \log(i)\right) + \frac{2 \cdot c_1}{N} + c_2 \stackrel{?}{\leq} c \cdot N \log(N)$$

The following left-hand-side is strictly bigger than the preceding.

$$\frac{2c}{N}\left(\sum_{i=1}^{N-1} i \log(N)\right) + \frac{2 \cdot c_1}{N} + c_2 \stackrel{?}{\leq} c \cdot N \log(N)$$

Quicksort's Expected Runtime

$$\frac{2c}{N}\left(\sum_{i=1}^{N-1} i\log(N)\right) + \frac{2\cdot c_1}{N} + c_2 \overset{?}{\leq} c\cdot N\log(N)$$

$$\frac{2c\log(N)}{N}\left(\sum_{i=1}^{N-1} i\right) + \frac{2\cdot c_1}{N} + c_2 \overset{?}{\leq} c\cdot N\log(N)$$

$$\frac{2c\log(N)}{N}\left(\frac{(N-1)\cdot N}{2}\right) + \frac{2\cdot c_1}{N} + c_2 \overset{?}{\leq} c\cdot N\log(N)$$

$$c\cdot N\log(N) - c\cdot\log(N) + \frac{2\cdot c_1}{N} + c_2 \overset{?}{\leq} c\cdot N\log(N)$$

$$\frac{2\cdot c_1}{N} + c_2 \overset{?}{\leq} c\cdot\log(N)$$

Quicksort's Expected Runtime

$$\frac{2 \cdot c_1}{N} + c_2 \stackrel{?}{\leq} c \cdot \log(N)$$

The following left-hand-side is strictly bigger than the preceding.

$$2 \cdot c_1 + c_2 \stackrel{?}{\leq} c \cdot \log(N)$$

True for any $N \geq 2$ if $c \geq 2 \cdot c_1 + c_2$

Quicksort's Expected Runtime

$$E[T_{qs}(N)] \in O(N \log(N))$$

So is Quicksort $O(N \log(N))$? **No!**

Quicksort's **Expected** runtime is $O(N \log(N))$

Bound Guarantees

- $f(N)$ is a Worst-Case Bound $(T(N) \in O(f(N)))$
  The algorithm **always** runs in at most $c \cdot f(N)$ steps.

- $f(N)$ is an Amortized Worst-Case Bound
  N **invocations** of the algorithm **always** run in at most $N \cdot c \cdot f(N)$ steps.

- $f(N)$ is an Expected Worst-Case Bound $(E[T(N)] \in O(f(N)))$
  The algorithm is **statistically likely** to run in at most $c \cdot f(N)$ steps.