Week 6 - Monday

COMP 2100

Last time

- What did we talk about last time?
- More recursion
- Exam post mortem

Questions?

Assignment 3

Recursion

Project 2

Infix to Postfix Converter

Maggie Smith 1934 - 2024

Back to Fibonacci

Code for better Fibonacci

```java
public static int fib2(int a, int b, int n) {

  if (n <= 2) {
     return b;
  } else {
     return fib2(b, a + b, n - 1);
  }
}

// proxy method
public static int fib(int n) {
  return fib2(1, 1, n);
}
```

Base Case

Recursive Case

Recursion for Exponentiation

Exponentiation

- We want to raise a number **x** to a power ***n***, like so: x^n
- We allow ***x*** to be real, but ***n*** must be an integer greater than or equal to 0
- Example: $(4.5)^{13} = 310286355.9971923828125$

Recursion for Exponentiation

- Base case ($n = 0$):
  - Result = 1
- Recursive case ($n > 0$):
  - Result = $x \cdot x^{(n-1)}$

Code for Exponentiation

```java
public static double power(double x, int n) {

  if (n == 0) {
     return 1;
  } else {
     return x * power(x, n - 1);
  }

}
```

Base Case

Recursive Case

Running time for `power`

- Each call reduces n by `1`
- n + `1` total calls
- What's the running time?
  - $\Theta(n)$

Can we do better than linear?

- We need to structure the recursion differently
- Instead of reducing n by 1 each time, can we reduce it by a lot more?
- It's true that $\boldsymbol{x^n} = \boldsymbol{x} \cdot \boldsymbol{x^{(n-1)}}$
- But, it is also true that $\boldsymbol{x^n} = \boldsymbol{x^{(n/2)}} \cdot \boldsymbol{x^{(n/2)}}$

New recursion for exponentiation

- Assume that n is a power of 2
- Base case ($n = 1$):
  - Result = x
- Recursive case ($n > 1$):
  - Result = $(x^{(n/2)})^2$

Code for better exponentiation

```java
public static double power2(double x, int n) {
  double temp;
  if (n == 1) {
     return x;
  } else {
     temp = power2( x, n/2 );
     return temp * temp;
  }
}
```

Base Case

Recursive Case

Running time for `power2`

- Each call reduces ***n*** by half
- $\log_2(n)$ total calls
- Just like binary search
- Can we expand the algorithm to even and odd values of ***n***?

Even newer recursion for exponentiation

- Base case ($n = 1$):
  - Result = x
- Recursive cases ($n > 1$):
  - If n is even, result = $(x^{(n/2)})^2$
  - If n is odd, result = $x \cdot (x^{((n-1)/2)})^2$

Code for Even Better Exponentiation

```
public static double power3( double x, int n )
{
  double temp;
  if( n == 1 )
     return x;
  else if( n % 2 == 0 )
  {
     temp = power3( x, n/2 );
     return temp * temp;
  }
  else
  {
     temp = power3( x, (n – 1)/2 );
     return x * temp * temp;
  }
}
```

Base Case

Recursive Cases

Running time for `power3`

- Each call reduces ***n*** by half (more or less)
- $\Theta(\log_2 n)$ total calls
- Does as well as **`power2()`**
- Better yet, we can use this solution to get a logarithmic time answer for Fibonacci!

- The ***n***th term of the Fibonacci sequence is:

$$\frac{\varphi^n - (1-\varphi)^n}{\sqrt{5}}$$

- Where $\varphi = \frac{1+\sqrt{5}}{2}$

Merge Sort

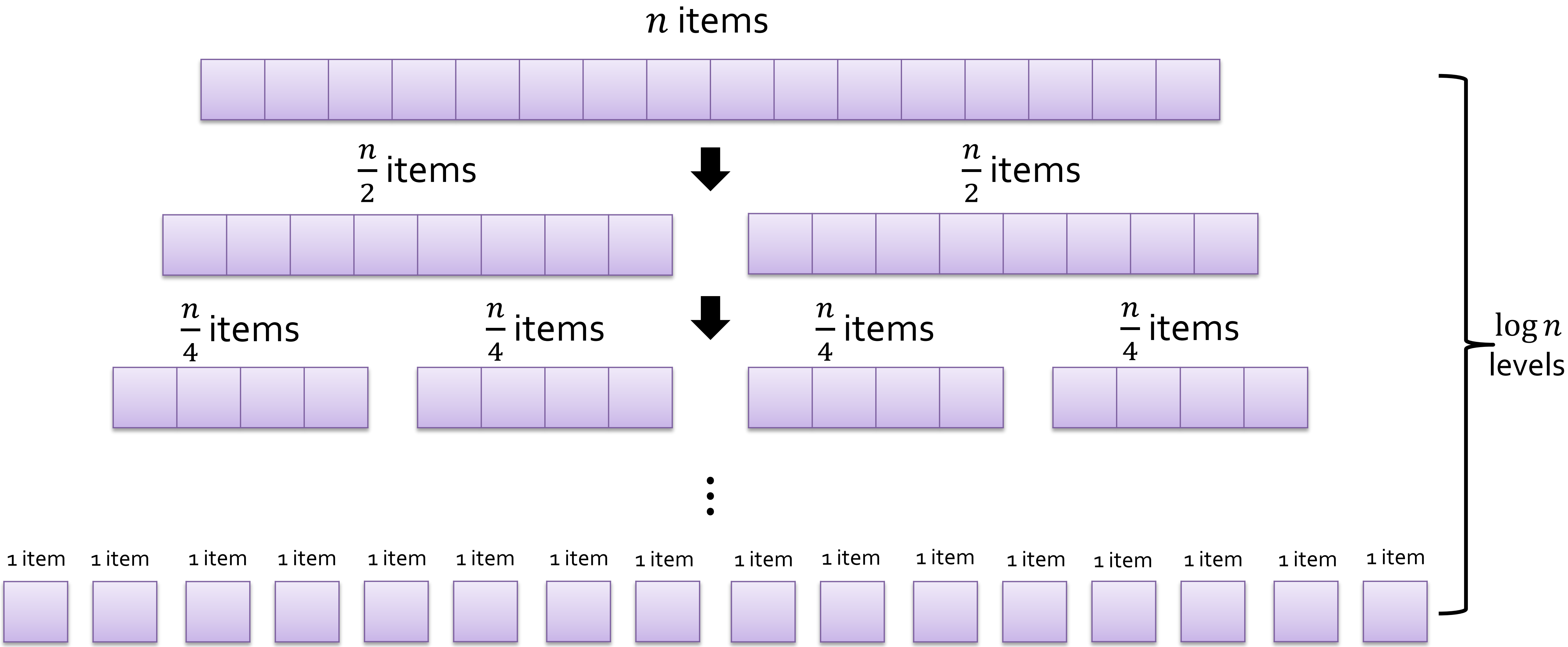
Merge Sort algorithm (recursive)

- Beautiful divide and conquer
- Base case: List has size 1
- Recursive case:
  - Divide your list in half
  - Recursively merge sort each half
  - Merge the two halves back together in sorted order

Let's code that up…

- Great. Now, how long does it take?

Running time for merge sort



Running time for merge sort

- At each level, $\Theta(n)$ work is done
  - Splitting up the array
  - Merging the array back
- There are $\log n$ levels
- Total running time is $\Theta(n \log n)$

Upcoming

Next time…

- Symbol tables
- Trees
- Binary search trees (BSTs)
- BST implementation

Reminders

- Work on Project 2
- Finish Assignment 3
  - Due this Friday
- Keep reading Section 3.2