Warm up

Show that finding the minimum of an unordered list requires \( \Omega(n) \) comparisons
Find Min, Lower Bound Proof

Show that finding the minimum of an unordered list requires $\Omega(n)$ comparisons

Suppose (toward contradiction) that there is an algorithm for Find Min that does fewer than $\frac{n}{2} = \Omega(n)$ comparisons.

This means there is at least one “uncompared” element. We can’t know that this element wasn’t the min!
Announcements

• HW4 due Monday 3/4 at 11pm
  – Sorting
  – Written (use LaTeX!)
• No Instructor Office Hours this week
  – I’ll be at SIGCSE
  – Available on Piazza and Email!
• HW1 solutions in-class on Wednesday
• Midterm next Wednesday
  – Covers material through today
  – Review session M or Tu evening
Today’s Keywords

• Sorting
• Linear time Sorting
• Counting Sort
• Radix Sort
• Maximum Sum Continuous Subarray
CLRS Readings

• Chapter 8
Sorting in Linear Time

• Cannot be comparison-based
• Need to make some sort of assumption about the contents of the list
  – Small number of unique values
  – Small range of values
  – Etc.
Counting Sort

• Idea: Count how many things are less than each element

Range is \([1, k]\) (here \([1,6]\))
make an array \(C\) of size \(k\)
populate with counts of each value

For \(i\) in \(L\):
\[
+ \ C[L[i]]
\]

1. Take “running sum” of \(C\)
to count things less than each value

For \(i = 1\) to \(\text{len}(C)\):
\[
C[i] = C[i-1] + C[i]
\]

To sort: last item of value 3 goes at index 4
Counting Sort

- **Idea:** Count how many things are less than each element

$L = 3 \ 6 \ 6 \ 1 \ 3 \ 4 \ 1 \ 6$

$C = 2 \ 2 \ 4 \ 5 \ 5 \ 7 \ 6 \ 8$

For each element of $L$ (last to first):

1. Use $C$ to find its proper place in $B$
2. Decrement that position of $C$

Last item of value 6 goes at index 8

For $i = \text{len}(L)$ downto 1:

\[
\begin{align*}
B[C[L[i]]] &= L[i] \\
C[L[i]] &= C[L[i]] - 1
\end{align*}
\]
Counting Sort

- **Idea:** Count how many things are less than each element

\[ L = \begin{bmatrix}
3 & 6 & 6 & 1 & 3 & 4 & 1 & 6 \\
1 & 2 & 3 & 4 & 5 & 6 & 7 & 8
\end{bmatrix} \]

\[ C = \begin{bmatrix}
1 & 2 & 4 & 5 & 5 & 7 \\
1 & 2 & 3 & 4 & 5 & 6
\end{bmatrix} \]

For each element of \( L \) (last to first):
- Use \( C \) to find its proper place in \( B \)
- Decrement that position of \( C \)

\[ B = \begin{bmatrix}
\_ & 1 & \_ & \_ & \_ & \_ & 6 \\
1 & 2 & 3 & 4 & 5 & 6 & 7 & 8
\end{bmatrix} \]

For \( i = \text{len}(L) \) downto 1:
\[
B \left[ C[L[i]] \right] = L[i] \\
C[L[i]] = C[L[i]] - 1
\]

- **Run Time:** \( O(n + k) \)
- **Memory:** \( O(n + k) \)
Counting Sort

• Why not always use counting sort?
• For 64-bit numbers, requires an array of length $2^{64} > 10^{19}$
  – 5 GHz CPU will require $> 116$ years to initialize the array
  – 18 Exabytes of data
    • Total amount of data that Google has
12 Exabytes
Radix Sort

• **Idea:** Stable sort on each digit, from least significant to most significant

<table>
<thead>
<tr>
<th>103</th>
<th>801</th>
<th>401</th>
<th>323</th>
<th>255</th>
<th>823</th>
<th>999</th>
<th>101</th>
<th>113</th>
<th>901</th>
<th>555</th>
<th>512</th>
<th>245</th>
<th>800</th>
<th>018</th>
<th>121</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>10</td>
<td>11</td>
<td>12</td>
<td>13</td>
<td>14</td>
<td>15</td>
</tr>
</tbody>
</table>

Place each element into a “bucket” according to its 1’s place

<table>
<thead>
<tr>
<th>800</th>
<th>801</th>
<th>401</th>
<th>101</th>
<th>901</th>
<th>121</th>
<th>512</th>
<th>103</th>
<th>323</th>
<th>823</th>
<th>113</th>
<th>255</th>
<th>555</th>
<th>245</th>
<th>018</th>
<th>999</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

12
Radix Sort

- **Idea:** *Stable sort* on each digit, from least significant to most significant

Place each element into a “bucket” according to its 10’s place

<table>
<thead>
<tr>
<th>800</th>
<th>801</th>
<th>103</th>
<th>121</th>
<th>512</th>
<th>323</th>
<th>255</th>
<th>245</th>
<th>018</th>
<th>999</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>800</th>
<th>801</th>
<th>512</th>
<th>113</th>
<th>018</th>
<th>121</th>
<th>245</th>
<th>255</th>
<th>555</th>
<th>999</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
</tr>
</tbody>
</table>
Radix Sort

- **Idea**: Stable sort on each digit, from least significant to most significant

Place each element into a “bucket” according to its 100’s place

Run Time: $O(d(n + b))$

$d = \text{digits in largest value}$

$b = \text{base of representation}$
Maximum Sum Continuous Subarray Problem

The maximum-sum subarray of a given array of integers $A$ is the interval $[a, b]$ such that the sum of all values in the array between $a$ and $b$ inclusive is maximal.

Given an array of $n$ integers (may include both positive and negative values), give a $O(n \log n)$ algorithm for finding the maximum-sum subarray.
Divide and Conquer $\Theta(n \log n)$
Divide and Conquer $\Theta(n \log n)$

- Divide in half
- Recursively Solve on Left
  19
- Recursively Solve on Right
  25
- Find Largest sum that spans the cut
- Largest sum that ends here
- Largest sum that starts here

```
5  8  -4  3  7  -15  2
```

```
8  -20  17  8  -50  -5  22
```

```
0  1  2  3  4  5  6  7  8  9  10  11  12  13
```

Divide and Conquer $\Theta(n \log n)$

Return the Max of 
Left, Right, Center

$T(n) = 2T\left(\frac{n}{2}\right) + n$
Divide and Conquer Summary

- **Divide**
  - Break the list in half
- **Conquer**
  - Find the best subarrays on the left and right
- **Combine**
  - Find the best subarray that “spans the divide”
  - I.e. the best subarray that ends at the divide concatenated with the best that starts at the divide

Typically multiple subproblems.
Typically all roughly the same size.
def myDCalgo(problem):
    if baseCase(problem):
        solution = solve(problem)  # brute force if necessary
        return solution
    subproblems = Divide(problem)
    for sub in subproblems:
        subsolutions.append(myDCalgo(sub))
    solution = Combine(subsolutions)
    return solution
MSCS Divide and Conquer $\Theta(n \log n)$

def MSCS(list):
    if list.length < 2:
        return list[0]  # list of size 1 the sum is maximal
    {listL, listR} = Divide(list)
    for list in {listL, listR}:
        subSolutions.append(MSCS(list))
    solution = max(solnL, solnR, span(listL, listR))
    return solution
Types of “Divide and Conquer”

• Divide and Conquer
  – Break the problem up into several subproblems of roughly equal size, recursively solve
  – E.g. Karatsuba, Closest Pair of Points, Mergesort...

• Decrease and Conquer
  – Break the problem into a single smaller subproblem, recursively solve
  – E.g. Gotham City Police, Quickselect, Binary Search
Pattern So Far

• Typically looking to divide the problem by some fraction (½, ¼ the size)

• Not necessarily always the best!
  – Sometimes, we can write faster algorithms by finding unbalanced divides.
Unbalanced Divide and Conquer

• **Divide**
  – Make a subproblem of all but the last element

• **Conquer**
  – Find best subarray on the left ($BSL(n - 1)$)
  – Find the best subarray ending at the divide ($BED(n - 1)$)

• **Combine**
  – New Best Ending at the Divide:
    • $BED(n) = \max(BED(n - 1) + arr[n], 0)$
  – New best on the left:
    • $BSL(n) = \max(BSL(n - 1), BED(n))$
Divide

Recursively
Solve on Left
25

Find Largest
sum ending at
the cut
22
Divide

Recursively
Solve on Left

25

Find Largest
sum ending at
the cut

0
Divide

Recursively
Solve on Left
25

Find Largest
sum ending at
the cut
0
Divide Recursively
Solve on Left
25

Find Largest sum ending at the cut
25
Divide
Recursively
Solve on Left

Find Largest
sum ending at
the cut
Divide

Recursively Solve on Left

Find Largest sum ending at the cut
Divide

Recursively Solve on Left

Find Largest sum ending at the cut
Unbalanced Divide and Conquer

• **Divide**
  – Make a subproblem of all but the last element

• **Conquer**
  – Find best subarray on the left \((BSL(n - 1))\)
  – Find the best subarray ending at the divide \((BED(n - 1))\)

• **Combine**
  – New Best Ending at the Divide:
    • \(BED(n) = \max(BED(n - 1) + arr[n], 0)\)
  – New best on the left:
    • \(BSL(n) = \max(BSL(n - 1), BED(n))\)
Was unbalanced better? **YES**

- **Old:**
  - We divided in **Half**
  - We solved 2 different problems:
    - Find the best overall on **BOTH** the left/right
    - Find the best which end/start on **BOTH** the left/right respectively
  - **Linear** time combine

- **New:**
  - We divide by **1, n-1**
  - We solve 2 different problems:
    - Find the best overall on the **left ONLY**
    - Find the best which ends on the **left ONLY**
  - **Constant** time combine

\[
T(n) = 2T\left(\frac{n}{2}\right) + n
\]

\[
T(n) = \Theta(n \log n)
\]

\[
T(n) = 1T(n - 1) + 1
\]

\[
T(n) = \Theta(n)
\]
Maximum Sum Continuous Subarray Problem Redux

- Solve in $O(n)$ by increasing the problem size by 1 each time.
- **Idea**: Only include negative values if the positives on both sides of it are “worth it”
Begin here

Remember two values:

- **Best So Far**
  - 5
- **Best ending here**
  - 5

\( \Theta(n) \) Solution
Θ(n) Solution

Remember two values: Best So Far
13
Best ending here
13
Θ(\(n\)) Solution

Remember two values:

- **Best So Far**: 13
- **Best ending here**: 9
$\Theta(n)$ Solution

Remember two values:

- Best So Far
  - 13

- Best ending here
  - 12
$\Theta(n)$ Solution

Remember two values:  
**Best So Far**

19  

**Best ending here**

19
$\Theta(n)$ Solution

Remember two values:

- **Best So Far**: 19
- **Best ending here**: 4
\( \Theta(n) \) Solution

Remember two values:

- **Best So Far**: 19
- **Best ending here**: 14
Θ(𝑛) Solution

Remember two values:
- **Best So Far**: 19
- **Best ending here**: 0
### $\Theta(n)$ Solution

<p>| | | | | | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>8</td>
<td>-4</td>
<td>3</td>
<td>7</td>
<td>-15</td>
<td>2</td>
<td>8</td>
<td>-20</td>
<td>17</td>
<td>8</td>
<td>-50</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>13</td>
</tr>
</tbody>
</table>

Remember two values:  

- **Best So Far**
  - 19  
- **Best ending here**
  - 17
Remember two values:

- **Best So Far**
  - 25

- **Best ending here**
  - 25
End of Midterm Exam Materials!

"Mr. Osborne, may I be excused? My brain is full."
Mid-Class Stretch

How many ways are there to tile a $2 \times n$ board with dominoes?

How many ways to tile this:  

With these?