Today’s agenda:
- Midterm solutions (on board)
- The Fastest Sorting Algorithms:
  - Mergesort
  - Quicksort

Covered in Chapter 7 of the textbook

Preorder Traversal with a Stack

```c
void Stack_Preorder (Tree T, Stack S)
{
    if (T == NULL) return; else push(T,S);
    while (!isempty(S)) {
        T = pop(S);
        print_element(T -> Element);
        if (T -> Right != NULL) push(T -> Right, S);
        if (T -> Left != NULL) push(T -> Left, S);
    }
}
```
Recall from Last Time: Mergesort

- Based on the idea of “Divide and Conquer”:
  1. Divide problem into smaller parts
  2. Independently solve the parts
  3. Combine these solutions to get overall solution

- **Mergesort**: Divide array into two halves, *recursively* sort left and right halves, then *merge* two halves

- Example: Mergesort the input array:

```
<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>2</td>
<td>9</td>
<td>4</td>
<td>5</td>
<td>3</td>
<td>1</td>
<td>6</td>
</tr>
</tbody>
</table>
```

Mergesort Example
Mergesort Analysis

- Let $T(N)$ be the running time for an array of $N$ elements
- Mergesort divides array in half and calls itself on the two halves. After returning, it merges both halves using a temporary array (see textbook for code).
- Each recursive call takes $T(N/2)$ and merging takes $O(N)$
- Therefore, the recurrence relation for $T(N)$ is:
  - $T(1) = O(1)$ (base case: 1 element array $\rightarrow$ constant time)
  - $T(N) = 2T(N/2) + N$

Solving the Mergesort Recurrence Relation

- Can solve the recurrence by expanding the terms:
  - $T(N) = 2^2T(N/2^2) + N$
  - Since $T(N/2) = 2^2T(N/4) + N/2$,
  - $T(N) = 2^2[2^2T(N/4) + N/2] + N$
    - $= 2^4T(N/2^2) + 2^2N$
    - $= 2^3[2^3T(N/8) + N/4] + 2^3N$
    - $= 2^3T(N/2^3) + 3N$
    - $\ldots$ (recall that $2^{\log N} = N$)
    - $= 2^{\log N}T(N/(2^{\log N})) + (\log N)^N$
    - $= \log N \cdot T(1) + N \log N$
    - $= N \cdot O(1) + N \log N = O(N \log N)$
  - $T(N) = O(N \log N)$
Quicksort

♦ Mergesort requires temporary array for merging \( \rightarrow O(N) \) extra space – can we do in place sorting without extra space?

♦ Quicksort also uses a divide and conquer strategy, but does not use the \( O(N) \) extra space

♦ Main Idea:
  ➤ Partition array into left and right sub-arrays
  ◆ Elements in left sub-array < elements in right sub-array
  ➤ Recursively sort left and right sub-arrays
  ➤ Concatenate left and right sub-arrays \( \rightarrow O(1) \) time operation

Partitioning in Quicksort

♦ Choose an element from the array as the pivot

♦ Move all elements < pivot into left sub-array and all elements > pivot into right sub-array
  ➤ The case where element = pivot can be handled in several ways

\[
\begin{array}{cccccc}
7 & 18 & 2 & 15 & 9 & 11 \\
\end{array}
\]

  ➤ Suppose pivot = 7
  ➤ Left subarray = 2
  ➤ Right sub-array = 18 15 9 11

♦ What is the running time for an array of \( N \) elements?
Quicksort Example

- Sort the array containing the elements:
  \[ 9 \ 12 \ 4 \ 15 \ 2 \ 5 \ 17 \ 1 \]

Questions to be Answered…

- How can we do partitioning in place?
- How do we pick the pivot to speed up running time?
- What is the best and worst case running time of Quicksort?
Answers? Next class – same place, same time…

To Do:
Read Chapter 7