DESIGN AND ANALYSIS OF ALGORITHMS

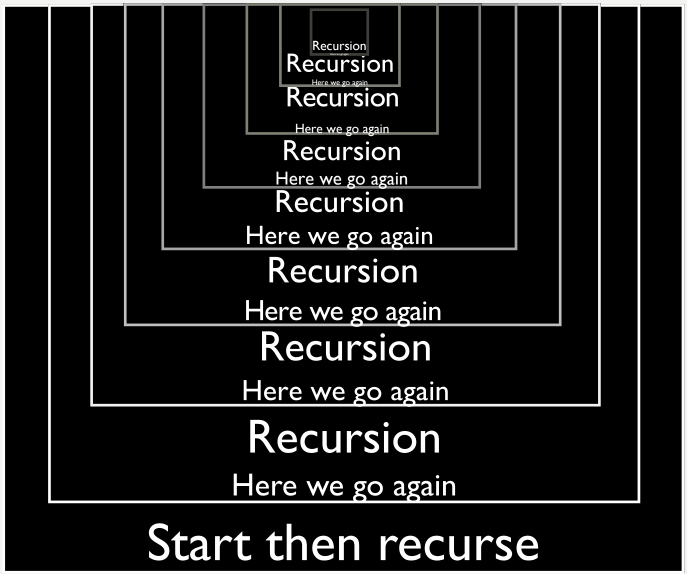
CS 4120/5120
DIVIDE AND CONQUER

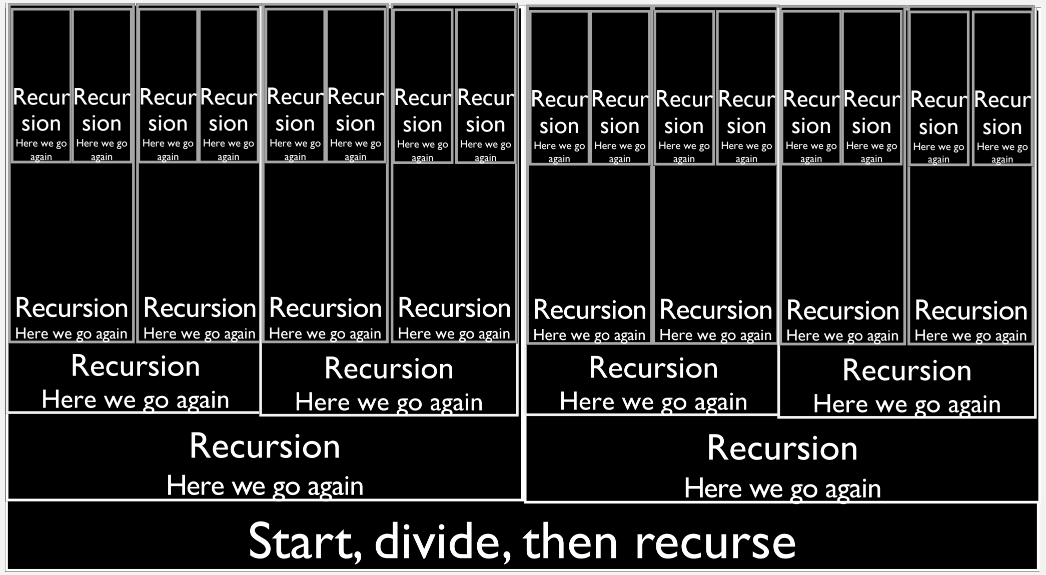
AGENDA

- Recursions
- Design techniques
- Divide and conquer
- Merge sort

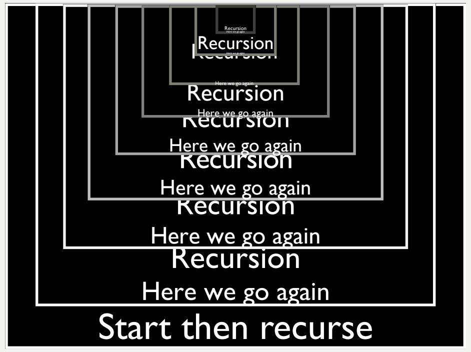
RECURSIONS

- The solution depends on solutions to smaller instances of the same problem.
- A recursive function calls itself recursively one or more times to deal with closely related





RECURSIONS
BINARY SEARCH



- Example: Given the following input sequence in increasing order and search for **78** in the input array.

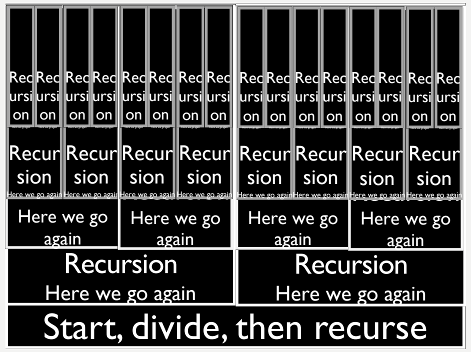
~~23~~ ~~45~~ ~~56~~ ~~65~~ | 83 | 86 | ~~93~~ ~~97~~

65 ∧ **78** — 83 ∨ **78** — 86 ∨ **78**

Not found

- Solution
  - Binary search
    - Cuts the input array in half. Search one half and discard the other half
- Each subproblem is closely related to the original problem.
- Solving the bottoms-out case means solving the entire problem.

RECURSIONS
MERGE SORT



- Example: Using merge sort to sort the input sequence below.
  - Input

    56 / 45 / 93 / 83 / 86 / 65 / 23 / 97
  - Merge 2

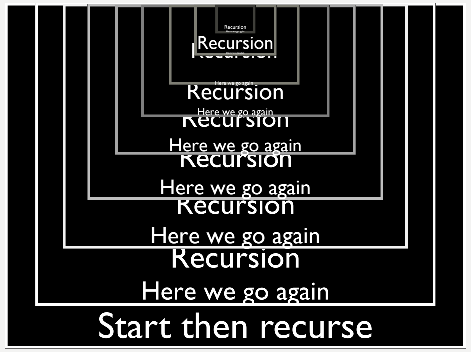
    45 56 83 93 65 86 23 97
  - Merge 4

    45 56 83 93 23 65 86 97
  - Merge 8 (final)

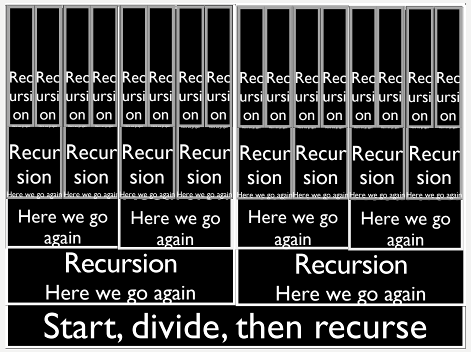
    23 45 45 65 83 86 93 97

TECHNIQUES THAT USE RECURSIONS

- Prune-and-Search
  - "Decrease-and-conquer"
  - The input size is reduced by a constant factor.
  - $T(n) = T(reduced\ n) + S(n)$



- Divide-and-Conquer
  - Divide the original problem
  - Combine solutions



DIVIDE-AND-CONQUER

- **Three steps**
  - **Divide** the problem into a number of subproblems that are smaller instances of the same problem.
  - **Conquer** the subproblems by solving them recursively.
    - If the subproblem sizes are small enough, just solve the subproblems in a straightforward manner.
  - **Combine** the solutions to the subproblems into the solution for the original problem.

- Algorithms
  - Merge sort, Maximum-subarray problem, Strassen's algorithm

MERGE SORT

- The MERGE-SORT algorithm

```
MERGE-SORT (A, p, r)
  if p < r
      q = ⌊(p + r)/2⌋
      MERGE-SORT (A, p, q)
      MERGE-SORT(A, q+1, r)
      MERGE (A, p, q, r)
```

- The problem is broken down into two subproblems, with each subproblem being half the original size.

MERGE SORT
RUNNING TIME ANALYSIS

- Complete the **cost-time** columns of the algorithm.
- For now, use function f to denote the running time of the MERGE algorithm.

$\boldsymbol{MERGE\text{-}SORT}(A, p, r)$	Cost	Time
if $p < r$	c_1	1
$\quad q = \lfloor (p + r)/2 \rfloor$	c_2	1
$\quad MERGE\text{-}SORT(A, p, q)$	$\boldsymbol{T(q - p + 1)}$	1
$\quad MERGE\text{-}SORT(A, q + 1, r)$	$\boldsymbol{T(r - q)}$	1
$\quad MERGE(A, p, q, r)$	$\boldsymbol{f(r - p + 1)}$	1

MERGE SORT
RUNNING TIME ANALYSIS

- The running time function.
  $T(r - p + 1)$
  $=$

$\boldsymbol{MERGE\text{-}SORT}(A, p, r)$		Cost	Time
1	**if** $p < r$	c_1	1
2	$q = \lfloor (p + r)/2 \rfloor$	c_2	1
3	$MERGE\text{-}SORT(A, p, q)$	$\boldsymbol{T(q - p + 1)}$	1
4	$MERGE\text{-}SORT(A, q + 1, r)$	$\boldsymbol{T(r - q)}$	1
5	$MERGE(A, p, q, r)$	$\boldsymbol{f(r - p + 1)}$	1

MERGE SORT
RUNNING TIME ANALYSIS

$\boldsymbol{MERGE\text{-}SORT}(A, p, r)$		Cost	Time
1	**if** $p < r$	c_1	1
2	$q = \lfloor (p + r)/2 \rfloor$	c_2	1
3	$MERGE\text{-}SORT(A, p, q)$	$\boldsymbol{T(q - p + 1)}$	1
4	$MERGE\text{-}SORT(A, q + 1, r)$	$\boldsymbol{T(r - q)}$	1
5	$MERGE(A, p, q, r)$	$\boldsymbol{f(r - p + 1)}$	1

- $T(r - p + 1)$
  $= \boldsymbol{c_1 + c_2 +}$
  $\boldsymbol{T(q - p + 1) +}$
  $\boldsymbol{T(r - q) +}$
  $\boldsymbol{f(\quad n \quad)}$
- Assume the input array A has n elements. To use MERGE-SORT algorithm, we can pass parameters ($\underline{\boldsymbol{A}}$, $\underline{\boldsymbol{1}}$, $\underline{\boldsymbol{n}}$).
- The relation of n, p, r is $n = \underline{\boldsymbol{r - p + 1}}$.
- Try to derive the running time in terms of n.

MERGE SORT
RUNNING TIME ANALYSIS

- $T(r - p + 1)$
  $= \boldsymbol{c_1 + c_2 + T(}q - p + 1\boldsymbol{) + T(r - q) + f(n)}$
- Plug q in $\boldsymbol{q - p + 1}$
  $\boldsymbol{q - p + 1}$
  $=$

$\boldsymbol{MERGE\text{-}SORT}(A, p, r)$	Cost	Time
if $p < r$	c_1	1
$q = \lfloor (p + r)/2 \rfloor$	c_2	1
$MERGE\text{-}SORT(A, p, q)$	$\boldsymbol{T(q - p + 1)}$	1
$MERGE\text{-}SORT(A, q + 1, r)$	$\boldsymbol{T(r - q)}$	1
$MERGE(A, p, q, r)$	$\boldsymbol{f(r - p + 1)}$	

MERGE SORT
RUNNING TIME ANALYSIS

- $T(r - p + 1)$
  $= \boldsymbol{c_1 + c_2 + T(}q - p + 1\boldsymbol{) +}$
  $\boldsymbol{T(}r - q\boldsymbol{) + f(n)}$

- Plug q in $\boldsymbol{r - q}$
  $\boldsymbol{r - q}$
  $=$

$\boldsymbol{MERGE\text{-}SORT}(A, p, r)$	Cost	Time
if $p < r$	c_1	1
$\quad q = \lfloor (p + r)/2 \rfloor$	c_2	1
$\quad q - p + 1$ $MERGE\text{-}SORT(A, p, q)$	$\boldsymbol{T(q - p + 1)}$	1
$\quad MERGE\text{-}SORT(A, q + 1, r)$	$\boldsymbol{T(r - q)}$	1
$\quad MERGE(A, p, q, r)$	$\boldsymbol{f(r - p + 1)}$	1

MERGE SORT
RUNNING TIME ANALYSIS

- $T(r - p + 1)$
  $= \boldsymbol{c_1 + c_2 + T(}q - p + 1\boldsymbol{) + T(}r - q\boldsymbol{) + f(n)}$

$\boldsymbol{MERGE\text{-}SORT}(A, p, r)$		Cost	Time
1	**if** $p < r$	c_1	1
2	$q = \lfloor (p + r)/2 \rfloor$	c_2	1
3	$MERGE\text{-}SORT(A, p, q)$	$\boldsymbol{T(q - p + 1)}$	1
4	$MERGE\text{-}SORT(A, q + 1, r)$	$\boldsymbol{T(r - q)}$	1
5	$MERGE(A, p, q, r)$	$\boldsymbol{f(r - p + 1)}$	1

- In summary

$$q - p + 1 = \begin{cases} & \text{(if } p + r \text{ is even)} \\ & \text{(if } p + r \text{ is odd)} \end{cases} \qquad r - q = \begin{cases} & \text{(if } p + r \text{ is even)} \\ & \text{(if } p + r \text{ is odd)} \end{cases}$$

MERGE SORT
RUNNING TIME ANALYSIS

- Derive the running time in terms of n.
  $T(n)$
  $=$

$\boldsymbol{MERGE\text{-}SORT}(A, p, r)$		Cost	Time
1	**if** $p < r$	c_1	1
2	$q = \lfloor (p + r)/2 \rfloor$	c_2	1
3	$MERGE\text{-}SORT(A, p, q)$	$\boldsymbol{T(q - p + 1)}$	1
4	$MERGE\text{-}SORT(A, q + 1, r)$	$\boldsymbol{T(r - q)}$	1
5	$MERGE(A, p, q, r)$	$\boldsymbol{f(r - p + 1)}$	1

(if $p + r$ is even)

$=$ (if $p + r$ is odd)

MERGE SORT
RUNNING TIME ANALYSIS

- Tolerate the sloppiness neglecting the $1/2$ in $n/2 \pm 1/2$.
- $T(n) = c_1 + c_2 + 2T\left(\frac{n}{2}\right) + f(n)$
- The function $T(n)$ is affected by the two recursions and MERGE.

$\boldsymbol{MERGE\text{-}SORT}(A, p, r)$		Cost	Time
1	**if** $p < r$	c_1	1
2	$q = \lfloor (p + r)/2 \rfloor$	c_2	1
3	$MERGE\text{-}SORT(A, p, q)$	$\boldsymbol{T(q - p + 1)}$	1
4	$MERGE\text{-}SORT(A, q + 1, r)$	$\boldsymbol{T(r - q)}$	1
5	$MERGE(A, p, q, r)$	$\boldsymbol{f(r - p + 1)}$	1

MERGE ANALYSIS

- Complete the ***cost*** and ***time*** columns.
  - Let $\boldsymbol{P}$ denote the probability of the condition in line 13 being true.

	$\boldsymbol{MERGE}(A, p, q, r)$	Cost	Time
1	$n_1 = q - p + 1$	c_1	1
2	$n_2 = r - q$	c_2	1
3	Let $L[1..n_1 + 1]$ and $R[1..n_2 + 1]$ be new arrays	c_3	1
4	**for** $i = 1$ **to** n_1	c_4	$n_1 + 1$
5	$\quad L[i] = A[p + i - 1]$	c_5	n_1
6	**for** $j = 1$ **to** n_2	c_6	$n_2 + 1$
7	$\quad R[j] = A[q + j]$	c_7	n_2
8	$L[n_1 + 1] = \infty$	c_8	1
9	$R[n_2 + 1] = \infty$	c_9	1
10	$i = 1$	c_{10}	1
11	$j = 1$	c_{11}	1
12	**for** $k = p$ **to** r	c_{12}	$(r - p) + 1 + 1$
13	$\quad$ **if** $L[i] \leq R[j]$	c_{13}	$(r - p) + 1$
14	$\quad\quad A[k] = L[i]$	c_{14}	$\boldsymbol{P}\big((r - p) + 1\big)$
15	$\quad\quad i = i + 1$	c_{15}	$\boldsymbol{P}\big((r - p) + 1\big)$
16	$\quad$ **else** $A[k] = R[j]$	c_{16}	$(1 - \boldsymbol{P})\big((r - p) + 1\big)$
17	$\quad\quad j = j + 1$	c_{17}	$(1 - \boldsymbol{P})\big((r - p) + 1\big)$

MERGE ANALYSIS

- Derive $f(r-p+1)$.

  Let $c = \max c_i$

  $f(r-p+1)$

  $=$

$\boldsymbol{MERGE}(A, p, q, r)$		Cost	Time
1	$n_1 = q - p + 1$	c_1	1
2	$n_2 = r - q$	c_2	1
3	Let $L[1..n_1 + 1]$ and $R[1..n_2 + 1]$ be new arrays	c_3	1
4	**for** $i = 1$ **to** n_1	c_4	$n_1 + 1$
5	$\quad L[i] = A[p + i - 1]$	c_5	n_1
6	**for** $j = 1$ **to** n_2	c_6	$n_2 + 1$
7	$\quad R[j] = A[q + j]$	c_7	n_2
8	$L[n_1 + 1] = \infty$	c_8	1
9	$R[n_2 + 1] = \infty$	c_9	1
10	$i = 1$	c_{10}	1
11	$j = 1$	c_{11}	1
12	**for** $k = p$ **to** r	c_{12}	$(r - p) + 1 + 1$
13	$\quad$ **if** $L[i] \leq R[j]$	c_{13}	$(r - p) + 1$
14	$\quad\quad A[k] = L[i]$	c_{14}	$\boldsymbol{P}\big((r - p) + 1\big)$
15	$\quad\quad i = i + 1$	c_{15}	$\boldsymbol{P}\big((r - p) + 1\big)$
16	$\quad$ **else** $A[k] = R[j]$	c_{16}	$(1 - \boldsymbol{P})\big((r - p) + 1\big)$
17	$\quad\quad j = j + 1$	c_{17}	$(1 - \boldsymbol{P})\big((r - p) + 1\big)$

MERGE ANALYSIS

- Assume the input A has n elements, $n =$ __________.
- The asymptotic tight bound of the MERGE in terms of n is $f(n) =$ ______________.

$\boldsymbol{MERGE}(A, p, q, r)$		Cost	Time
1	$n_1 = q - p + 1$	c_1	1
2	$n_2 = r - q$	c_2	1
3	Let $L[1..n_1 + 1]$ and $R[1..n_2 + 1]$ be new arrays	c_3	1
4	**for** $i = 1$ **to** n_1	c_4	$\boldsymbol{n_1} + 1$
5	$L[i] = A[p + i - 1]$	c_5	$\boldsymbol{n_1}$
6	**for** $j = 1$ **to** n_2	c_6	$\boldsymbol{n_2} + 1$
7	$R[j] = A[q + j]$	c_7	$\boldsymbol{n_2}$
8	$L[n_1 + 1] = \infty$	c_8	1
9	$R[n_2 + 1] = \infty$	c_9	1
10	$i = 1$	c_{10}	1
11	$j = 1$	c_{11}	1
12	**for** $k = p$ **to** r	c_{12}	$(r - p) + 1 + 1$
13	**if** $L[i] \le R[j]$	c_{13}	$(r - p) + 1$
14	$A[k] = L[i]$	c_{14}	$\boldsymbol{P}\big((r - p) + 1\big)$
15	$i = i + 1$	c_{15}	$\boldsymbol{P}\big((r - p) + 1\big)$
16	**else** $A[k] = R[j]$	c_{16}	$(1 - \boldsymbol{P})\big((r - p) + 1\big)$
17	$j = j + 1$	c_{17}	$(1 - \boldsymbol{P})\big((r - p) + 1\big)$

MERGE SORT
TIME COMPLEXITY

- In conclusion, the running time of MERGE-SORT is

$$T(n) = \begin{cases} \Theta(1), & n = 1 \\ 2T\left(\frac{n}{2}\right) + \Theta(n), & n > 1 \end{cases}$$

- The **bottoms-out case** is when the array has only one number, which can be considered as sorted.
- Bound the function: $T(n) = \Theta(n \log n)$ or $T(n) = O(n \log n)$.
  - We will see some bounding techniques in future classes.

NEXT UP
THE MAXIMUM-SUBARRAY PROB.

REFERENCE

- https://en.wikipedia.org/wiki/Recursion_(computer_science)