CS 2500: Algorithms

Lecture 17: Divide-and-Conquer: Median and Order Statistics

Shubham Chatterjee

Missouri University of Science and Technology, Department of Computer Science

October 15, 2024

Order Statistics: Definition

- The i-**th order statistic** of a set of n elements is the i-th smallest element.
- **Examples:**
  - The **minimum** of a set is the first order statistic ($i = 1$).
  - The **maximum** of a set is the n-th order statistic ($i = n$).
- A **median** is the "halfway point" of the set:
  - If n is odd, the median occurs at $i = \frac{n+1}{2}$.
  - If n is even, two medians exist:
    - **Lower median**: $i = \frac{n}{2}$
    - **Upper median**: $i = \frac{n}{2} + 1$

Order Statistics: Median Simplification

- For simplicity, we refer to the **lower median** as **the median**.
- Medians occur at:

$$i = \left\lfloor \frac{n+1}{2} \right\rfloor \quad \text{and} \quad i = \left\lceil \frac{n+1}{2} \right\rceil$$

The Selection Problem: Definition

- **Goal:** Select the i-th order statistic from a set of n distinct numbers.
- **Input:**
  - A set A of n distinct numbers.
  - An integer i, where $1 \leq i \leq n$.
- **Output:** The element $x \in A$ that is larger than exactly $i - 1$ other elements of A.

Naive Solution to the Selection Problem

- Sort the set A using **Heapsort** or **Merge Sort** in $O(n \log n)$ time.
- Output the i-th element from the sorted array.
- **Drawback:** Sorting requires $O(n \log n)$ time, but we can solve the selection problem **asymptotically faster**.

Selecting the Minimum and Maximum

- This problem involves finding both the **maximum and minimum** elements in a set of n elements.
- We use a **divide-and-conquer** approach to solve this problem efficiently.
- The main focus is on the **number of element comparisons**, since:
  - The frequency of other operations is similar to element comparisons.
  - Comparisons dominate when elements are complex (e.g., large numbers, strings).

Selecting the Minimum and Maximum: Straightforward Approach

StraightMaxMin(a, n, max, min)

```
max ← a[1]
min ← a[1]
for i ← 2 to n do
    if a[i] > max then
        max ← a[i]
    end if
    if a[i] < min then
        min ← a[i]
    end if
end for
```

Time Complexity Analysis

- **Best case:** Elements in increasing order, requiring $n-1$ comparisons.
- **Worst case:** Elements in decreasing order, requiring $2(n-1)$ comparisons.
- **Average case:** $< 2(n-1)$ comparisons.

Selecting the Minimum and Maximum: Divide-and-Conquer Approach

- We use a **divide-and-conquer** strategy:
  - Divide the list into smaller sublists.
  - Recursively solve for each sublist.
  - Combine the results to get the final solution.

Selecting the Minimum and Maximum: Divide-and-Conquer Approach

Base Case: Small Inputs

- If the list has **one element** ($n = 1$):
  - Both the maximum and minimum are that single element.
- If the list has **two elements** ($n = 2$):
  - One comparison is needed to find the max and min.

Selecting the Minimum and Maximum: Divide-and-Conquer Approach

Divide Step: Partitioning the List

- If the list has more than two elements, divide it into two parts:

$$P_1 = \left(a[1], \ldots, a\left[\frac{n}{2}\right]\right) \quad \text{and} \quad P_2 = \left(a\left[\frac{n}{2}+1\right], \ldots, a[n]\right)$$

- Recursively apply the divide-and-conquer algorithm to both parts.

Selecting the Minimum and Maximum: Divide-and-Conquer Approach

Conquer Step: Combining Results

- After finding the maximum and minimum for P_1 and P_2:
  - **MAX(P)** is the larger of:

$$\max\left(\mathrm{MAX}(P_1), \mathrm{MAX}(P_2)\right)$$

  - **MIN(P)** is the smaller of:

$$\min\left(\mathrm{MIN}(P_1), \mathrm{MIN}(P_2)\right)$$

- The final max and min are determined from the two sublists.

Selecting the Minimum and Maximum: Divide-and-Conquer Approach

```
Algorithm MaxMin(i, j, max, min)
// a[1 : n] is a global array. Parameters i and j are integers,
// 1 ≤ i ≤ j ≤ n. The effect is to set max and min to the
// largest and smallest values in a[i : j], respectively.
{
    if (i = j) then max := min := a[i]; // Small(P)
    else if (i = j − 1) then  // Another case of Small(P)
        {
            if (a[i] < a[j]) then
            {
                max := a[j]; min := a[i];
            }
            else
            {
                max := a[i]; min := a[j];
            }
        }
        else
        {   // If P is not small, divide P into subproblems.
            // Find where to split the set.
                mid := ⌊(i + j)/2⌋;
            // Solve the subproblems.
                MaxMin(i, mid, max, min);
                MaxMin(mid + 1, j, max1, min1);
            // Combine the solutions.
                if (max < max1) then max := max1;
                if (min > min1) then min := min1;
        }
}
```

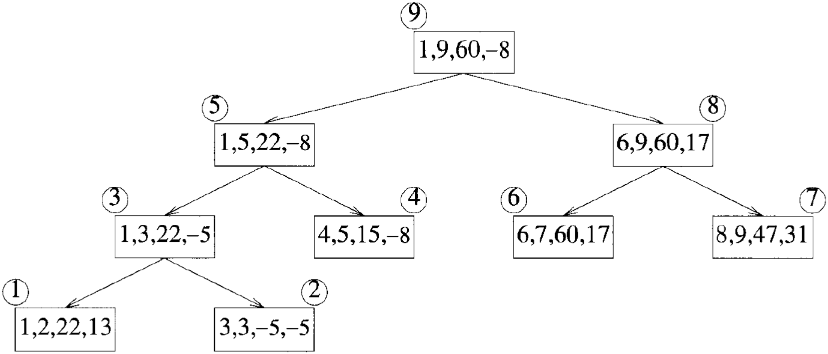
Selecting the Minimum and Maximum: Divide-and-Conquer Approach

Example: Show how the divide-and-conquer approach `MaxMin` works for the following set of number:

a:	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
	22	13	-5	-8	15	60	17	31	47

Selecting the Minimum and Maximum: Divide-and-Conquer Approach

Example: Show how the divide-and-conquer approach `MaxMin` works for the following set of number:



Selecting the Minimum and Maximum: Divide-and-Conquer Approach

Recurrence Relation for MaxMin

- The number of element comparisons required by the MaxMin algorithm follows the recurrence relation:

$$T(n) = \begin{cases} T\left(\left\lceil \frac{n}{2} \right\rceil\right) + T\left(\left\lfloor \frac{n}{2} \right\rfloor\right) + 2 & \text{if } n > 2 \\ 1 & \text{if } n = 2 \\ 0 & \text{if } n = 1 \end{cases}$$

Selecting the Minimum and Maximum: Divide-and-Conquer Approach

Solving the recurrence

- Assume $n = 2^k$ for some positive integer k.
- Expanding the recurrence relation step by step:

$$
\begin{aligned}
T(n) &= 2T\left(\frac{n}{2}\right) + 2 \\
&= 2\left(2T\left(\frac{n}{4}\right) + 2\right) + 2 = 4T\left(\frac{n}{4}\right) + 4 + 2 \\
&\vdots \\
&= 2^{k-1}T(2) + \sum_{i=1}^{k-1} 2^i
\end{aligned}
$$

- Substituting $T(2) = 1$:

$$T(n) = 2^{k-1} \cdot 1 + \sum_{i=1}^{k-1} 2^i$$

Selecting the Minimum and Maximum: Divide-and-Conquer Approach

Solving the recurrence

- In the `MaxMin` analysis, we encounter the following geometric series:

$$\sum_{i=1}^{k-1} 2^i = 2^1 + 2^2 + 2^3 + \ldots + 2^{k-1}$$

- The sum of the first n terms of a geometric series is:

$$S_n = a\frac{r^n - 1}{r - 1}$$

- For our series:
    - $a = 2$ (the first term).
    - $r = 2$ (the common ratio).
    - $n = k - 1$ (number of terms).
- Substituting into the formula:

$$S = 2\frac{2^{k-1} - 1}{2 - 1} = 2(2^{k-1} - 1) = 2^k - 2$$

Selecting the Minimum and Maximum: Divide-and-Conquer Approach

Solving the recurrence

- $2^{k-1}T(2) + \sum_{i=1}^{k-1} 2^i$
- Substituting $T(2) = 1$ and $\sum_{i=1}^{k-1} 2^i = 2^k - 2$

$$T(n) = 2^{k-1} + 2^k - 2$$

- As $n = 2^k$, we get:

$$T(n) = \frac{n}{2} + n - 2 = \frac{3n}{2} - 2$$

Selecting the Minimum and Maximum: Divide-and-Conquer Approach

Comparison with Straightforward Method

- For the straightforward method, the number of comparisons is:

$$2n - 2$$

- For the divide-and-conquer algorithm, the number of comparisons is:

$$\frac{3n}{2} - 2$$

- This represents a savings of approximately **25%** in comparisons.