# Algorithms 2021: Searching and Sorting

(Based on [Manber 1989])

Yih-Kuen Tsay

October 26, 2021

## 1 Binary Search

### Searching a Sorted Sequence

**Problem 1.** *Let $x_1, x_2, \cdots, x_n$ be a sequence of real numbers such that $x_1 \leq x_2 \leq \cdots \leq x_n$. Given a real number $z$, we want to find whether $z$ appears in the sequence, and, if it does, to find an index $i$ such that $x_i = z$.*

Idea: cut the search space in half by asking only one question.

$$\begin{cases} T(1) = O(1) \\ T(n) = T(\frac{n}{2}) + O(1), n \geq 2 \end{cases}$$

Time complexity: $O(\log n)$ (applying the master theorem with $a = 1$, $b = 2$, $k = 0$, and $b^k = 1 = a$).

### Binary Search

**function Find** $(z, Left, Right) : integer$;
**begin**
  **if** $Left = Right$ **then**
    **if** $X[Left] = z$ **then** $Find := Left$
    **else** $Find := 0$
  **else**
    $Middle := \lceil \frac{Left+Right}{2} \rceil$;
    **if** $z < X[Middle]$ **then**
      $Find := Find(z, Left, Middle - 1)$
    **else**
      $Find := Find(z, Middle, Right)$
**end**

**Algorithm Binary_Search** $(X, n, z)$;
**begin**
  $Position := Find(z, 1, n)$;
**end**

### Binary Search: Alternative

**function Find** $(z, Left, Right) : integer$;
**begin**
  **if** $Left > Right$ **then**
    $Find := 0$
  **else**
    $Middle := \lceil \frac{Left+Right}{2} \rceil$;
    **if** $z = X[Middle]$ **then**
      $Find := Middle$
    **else if** $z < X[Middle]$ **then**
      $Find := Find(z, Left, Middle - 1)$
    **else**
      $Find := Find(z, Middle + 1, Right)$
**end**

How do the two algorithms compare?

## 1.1 Cyclically Sorted Sequence

### Searching a Cyclically Sorted Sequence

**Problem 2.** *Given a cyclically sorted list, find the position of the minimal element in the list (we assume, for simplicity, that this position is unique).*

- Example 1:

  | | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | |
  |---|---|---|---|---|---|---|---|---|---|
  | [ | 5 | 6 | 7 | 0 | 1 | 2 | 3 | 4 | ] |

  - The 4th is the minimal element.

- Example 2:

  | | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | |
  |---|---|---|---|---|---|---|---|---|---|
  | [ | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | ] |

  - The 1st is the minimal element.

- To cut the search space in half, what question should we ask?

  /* If $X[Middle] < X[Right]$, then the minimal is in the left half (including $X[Middle]$; otherwise, it is in the right half (excluding $X[Middle]$). */

### Cyclic Binary Search

**Algorithm Cyclic_Binary_Search** $(X, n)$;
**begin**
  $Position := Cyclic\_Find(1, n)$;
**end**

**function Cyclic_Find** $(Left, Right) : integer$;
**begin**
  **if** $Left = Right$ **then** $Cyclic\_Find := Left$
  **else**
    $Middle := \lfloor \frac{Left+Right}{2} \rfloor$;
    **if** $X[Middle] < X[Right]$ **then**
      $Cyclic\_Find := Cyclic\_Find(Left, Middle)$

```
        else
            Cyclic_Find := Cyclic_Find(Middle + 1, Right)
end
```

## 1.2 "Fixpoints"

**"Fixpoints"**

**Problem 3.** *Given a sorted sequence of* distinct *integers* $a_1, a_2, \cdots, a_n$*, determine whether there exists an index* $i$ *such that* $a_i = i$*.*

- Example 1:

  | | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
  |---|---|---|---|---|---|---|---|---|
  | [ | −1 | 1 | 2 | 4 | 5 | 6 | 8 | 9 ] |

  - $a_4 = 4$ (there are more ...).

- Example 2:

  | | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
  |---|---|---|---|---|---|---|---|---|
  | [ | −1 | 1 | 2 | 5 | 6 | 8 | 9 | 10 ] |

  - There is no $i$ such that $a_i = i$.

- Again, can we cut the search space in half by asking only one question?

  /* As the numbers are distinct, they increase or decrease at least as fast as the indices (which always increase or decrease by one). If $X[Middle] < Middle$, then the fixpoint (if it exists) must be in the left half (excluding $X[Middle]$); otherwise, it must be in the right half (including $X[Middle]$). */

**A Special Binary Search**

```
function Special_Find (Left, Right) : integer;
begin
    if Left = Right then
        if A[Left] = Left then Special_Find := Left
        else Special_Find := 0
    else
        Middle := ⌊(Left+Right)/2⌋;
        if A[Middle] < Middle then
            Special_Find := Special_Find(Middle + 1, Right)
        else
            Special_Find := Special_Find(Left, Middle)
end
```

**A Special Binary Search (cont.)**

```
Algorithm Special_Binary_Search (A, n);
begin
    Position := Special_Find(1, n);
end
```

## 1.3 Stuttering Subsequence

**Stuttering Subsequence**

**Problem 4.** *Given two sequences $A$ ($= a_1 a_2 \cdots a_n$) and $B$ ($= b_1 b_2 \cdots b_m$), find the maximal value of $i$ such that $B^i$ is a subsequence of $A$.*

- If $B = xyzzx$, then $B^2 = xxyyzzzzxx$, $B^3 = xxxyyyzzzzzzxxx$, etc.
- $B$ is a subsequence of $A$ if we can embed $B$ inside $A$ in the same order but with possible holes.
- For example, $B^2 = xxyyzzzzxx$ is a subsequence of $xxzzyyyyxxzzzzzxxx$.
- If $B^j$ is a subsequence of $A$, then $B^i$ is a subsequence of $A$, for $1 \leq i \leq j$.
- The maximum value of $i$ cannot exceed $\lfloor \frac{n}{m} \rfloor$ (or $B^i$ would be longer than $A$).

**Stuttering Subsequence (cont.)**

Two ways to find the maximum $i$:

- Sequential search: try 1, 2, 3, etc. sequentially.
  Time complexity: $O(nj)$, where $j$ is the maximum value of $i$.
- Binary search between 1 and $\lfloor \frac{n}{m} \rfloor$.
  Time complexity: $O(n \log \frac{n}{m})$.

Can binary search be applied, if the bound $\lfloor \frac{n}{m} \rfloor$ is unknown?

Think of the base case in a reversed induction.

/* Try $2^0$, $2^1$, $2^2$, $\cdots$, $2^{k-1}$, and $2^k$ sequentially. If the target falls between $2^{k-1}$ and $2^k$, apply binary search within that region. */

# 2 Interpolation Search

**Interpolation Search**

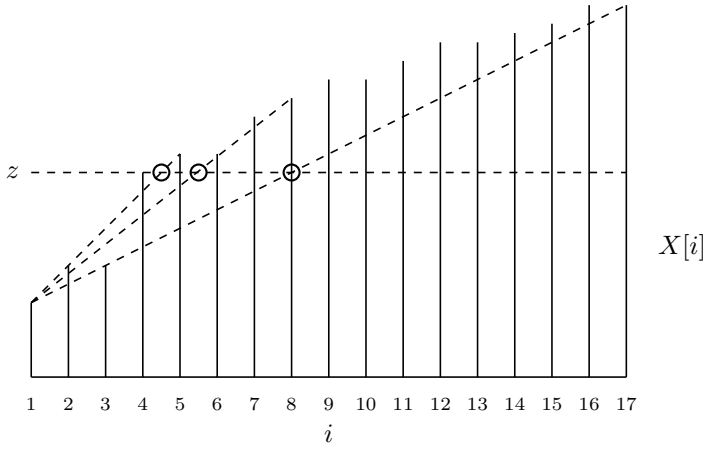
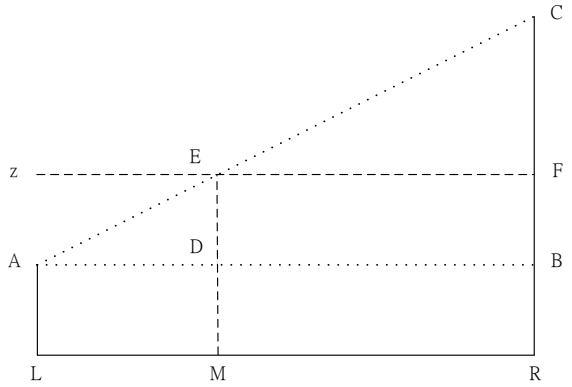



Figure: Interpolation search.
Source: redrawn from [Manber 1989, Figure 6.4].

**Interpolation Search (cont.)**




$$\frac{\overline{LM}}{\overline{LR}} = \frac{\overline{AD}}{\overline{AB}} = \frac{\overline{AE}}{\overline{AC}} = \frac{\overline{BF}}{\overline{BC}}, \text{so } |\overline{LM}| = \frac{|\overline{BF}|}{|\overline{BC}|} \times |\overline{LR}|$$

**Interpolation Search (cont.)**

**function Int_Find** $(z, Left, Right) : integer$;
**begin**
  **if** $X[Left] = z$ **then** $Int\_Find := Left$
  **else if** $Left = Right$ or $X[Left] = X[Right]$ **then**
    $Int\_Find := 0$
  **else**
    $Next\_Guess := \lceil Left + \frac{(z - X[Left])(Right - Left)}{X[Right] - X[Left]} \rceil$;
    **if** $z < X[Next\_Guess]$ **then**
      $Int\_Find := Int\_Find(z, Left, Next\_Guess - 1)$
    **else**
      $Int\_Find := Int\_Find(z, Next\_Guess, Right)$
**end**

/* $Next\_Guess - Left = |\overline{LM}| = \frac{|\overline{BF}|}{|\overline{BC}|} \times |\overline{LR}| \approx \lceil \frac{(z - X[Left])(Right - Left)}{X[Right] - X[Left]} \rceil$ */

**Interpolation Search (cont.)**

**Algorithm Interpolation_Search** $(X, n, z)$;
**begin**
  **if** $z < X[1]$ or $z > X[n]$ **then** $Position := 0$
  **else** $Position := Int\_Find(z, 1, n)$;
**end**

# 3 Sorting

**Sorting**

**Problem 5.** *Given $n$ numbers $x_1$, $x_2$, $\cdots$, $x_n$, arrange them in increasing order. In other words, find a sequence of distinct indices $1 \le i_1, i_2, \cdots, i_n \le n$, such that $x_{i_1} \le x_{i_2} \le \cdots \le x_{i_n}$.*

A sorting algorithm is called **in-place** if no additional work space is used besides the initial array that holds the elements.

## 3.1 Using Balanced Search Trees

**Using Balanced Search Trees**

- Balanced search trees, such as AVL trees, may be used for sorting:
  1. Create an empty tree.
  2. Insert the numbers one by one to the tree.
  3. Traverse the tree and output the numbers.
- What's the time complexity? Suppose we use an AVL tree.

## 3.2 Radix Sort

**Radix Sort**

**Algorithm Straight_Radix** $(X, n, k)$;
**begin**
    *put all elements of $X$ in a queue $GQ$*;
    **for** $i := 1$ **to** $d$ **do**
        *initialize queue $Q[i]$ to be empty*
    **for** $i := k$ **downto** 1 **do**
        **while** *$GQ$ is not empty* **do**
            *pop $x$ from $GQ$*;
            *$d :=$ the $i$-th digit of $x$*;
            *insert $x$ into $Q[d]$*;
        **for** $t := 1$ **to** $d$ **do**
            *insert $Q[t]$ into $GQ$*;
    **for** $i := 1$ **to** $n$ **do**
        *pop $X[i]$ from $GQ$*
**end**

Time complexity: $O(nk)$.

## 3.3 Merge Sort

**Merge Sort**

**Algorithm Mergesort** $(X, n)$;
**begin** $M\_Sort(1, n)$ **end**

**procedure M_Sort** $(Left, Right)$;
**begin**
    **if** $Right - Left = 1$ **then**
      **if** $X[Left] > X[Right]$ **then** $swap(X[Left], X[Right])$
    **else if** $Left \neq Right$ **then**
        $Middle := \lceil \frac{1}{2}(Left + Right) \rceil$;
        $M\_Sort(Left, Middle - 1)$;
        $M\_Sort(Middle, Right)$;

**Merge Sort (cont.)**

```
        i := Left;  j := Middle;  k := 0;
        while (i ≤ Middle − 1) and (j ≤ Right) do
              k := k + 1;
              if X[i] ≤ X[j] then
                 TEMP[k] := X[i];  i := i + 1
              else TEMP[k] := X[j];  j := j + 1;
        if j > Right then
           for t := 0 to Middle − 1 − i do
              X[Right − t] := X[Middle − 1 − t]
        for t := 0 to k − 1 do
           X[Left + t] := TEMP[1 + t]
end
```

Time complexity: $O(n \log n)$.

**Merge Sort (cont.)**

| | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 6 | 2 | 8 | 5 | 10 | 9 | 12 | 1 | 15 | 7 | 3 | 13 | 4 | 11 | 16 | 14 |
| ② | ⑥ | 8 | 5 | 10 | 9 | 12 | 1 | 15 | 7 | 3 | 13 | 4 | 11 | 16 | 14 |
| 2 | 6 | ⑤ | ⑧ | 10 | 9 | 12 | 1 | 15 | 7 | 3 | 13 | 4 | 11 | 16 | 14 |
| ② | ⑤ | ⑥ | ⑧ | 10 | 9 | 12 | 1 | 15 | 7 | 3 | 13 | 4 | 11 | 16 | 14 |
| 2 | 5 | 6 | 8 | ⑨ | ⑩ | 12 | 1 | 15 | 7 | 3 | 13 | 4 | 11 | 16 | 14 |
| 2 | 5 | 6 | 8 | 9 | 10 | ① | ⑫ | 15 | 7 | 3 | 13 | 4 | 11 | 16 | 14 |
| 2 | 5 | 6 | 8 | ① | ⑨ | ⑩ | ⑫ | 15 | 7 | 3 | 13 | 4 | 11 | 16 | 14 |
| ① | ② | ⑤ | ⑥ | ⑧ | ⑨ | ⑩ | ⑫ | 15 | 7 | 3 | 13 | 4 | 11 | 16 | 14 |
| 1 | 2 | 5 | 6 | 8 | 9 | 10 | 12 | ⑦ | ⑮ | 3 | 13 | 4 | 11 | 16 | 14 |
| 1 | 2 | 5 | 6 | 8 | 9 | 10 | 12 | 7 | 15 | ③ | ⑬ | 4 | 11 | 16 | 14 |
| 1 | 2 | 5 | 6 | 8 | 9 | 10 | 12 | ③ | ⑦ | ⑬ | ⑮ | 4 | 11 | 16 | 14 |
| 1 | 2 | 5 | 6 | 8 | 9 | 10 | 12 | 3 | 7 | 13 | 15 | ④ | ⑪ | 16 | 14 |
| 1 | 2 | 5 | 6 | 8 | 9 | 10 | 12 | 3 | 7 | 13 | 15 | 4 | 11 | ⑭ | ⑯ |
| 1 | 2 | 5 | 6 | 8 | 9 | 10 | 12 | 3 | 7 | 13 | 15 | ④ | ⑪ | ⑭ | ⑯ |
| 1 | 2 | 5 | 6 | 8 | 9 | 10 | 12 | ③ | ④ | ⑦ | ⑪ | ⑬ | ⑭ | ⑮ | ⑯ |
| ① | ② | ③ | ④ | ⑤ | ⑥ | ⑦ | ⑧ | ⑨ | ⑩ | ⑪ | ⑫ | ⑬ | ⑭ | ⑮ | ⑯ |

Figure: An example of mergesort.
Source: redrawn from [Manber 1989, Figure 6.8].

## 3.4 Quick Sort

**Quick Sort**

```
Algorithm Quicksort (X, n);
begin
    Q_Sort(1, n)
end

procedure Q_Sort (Left, Right);
begin
    if Left < Right then
```

$Partition(X, Left, Right)$;
$Q\_Sort(Left, Middle - 1)$;
$Q\_Sort(Middle + 1, Right)$
**end**

Time complexity: $O(n^2)$, but $O(n \log n)$ in average

**Quick Sort (cont.)**

**Algorithm Partition**$(X, Left, Right)$;
**begin**
$pivot := X[Left]$;
$L := Left$; $R := Right$;
**while** $L < R$ **do**
**while** $X[L] \leq pivot$ and $L \leq Right$ **do** $L := L + 1$;
**while** $X[R] > pivot$ and $R \geq Left$ **do** $R := R - 1$;
**if** $L < R$ **then** $swap(X[L], X[R])$;
$Middle := R$;
$swap(X[Left], X[Middle])$
**end**

**Quick Sort (cont.)**

| | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 6 | 2 | 8 | 5 | 10 | 9 | 12 | 1 | 15 | 7 | 3 | 13 | 4 | 11 | 16 | 14 |
| 6 | 2 | ④ | 5 | 10 | 9 | 12 | 1 | 15 | 7 | 3 | 13 | ⑧ | 11 | 16 | 14 |
| 6 | 2 | 4 | 5 | ③ | 9 | 12 | 1 | 15 | 7 | ⑩ | 13 | 8 | 11 | 16 | 14 |
| 6 | 2 | 4 | 5 | 3 | ① | 12 | ⑨ | 15 | 7 | 10 | 13 | 8 | 11 | 16 | 14 |
| ① | 2 | 4 | 5 | 3 | ⑥ | 12 | 9 | 15 | 7 | 10 | 13 | 8 | 11 | 16 | 14 |

Figure: Partition of an array around the pivot 6.
Source: redrawn from [Manber 1989, Figure 6.10].

**Quick Sort (cont.)**

| | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 6 | 2 | 8 | 5 | 10 | 9 | 12 | 1 | 15 | 7 | 3 | 13 | 4 | 11 | 16 | 14 |
| 1 | 2 | 4 | 5 | 3 | ⑥ | 12 | 9 | 15 | 7 | 10 | 13 | 8 | 11 | 16 | 14 |
| ① | 2 | 4 | 5 | 3 | ⑥ | 12 | 9 | 15 | 7 | 10 | 13 | 8 | 11 | 16 | 14 |
| ① | ② | 4 | 5 | 3 | ⑥ | 12 | 9 | 15 | 7 | 10 | 13 | 8 | 11 | 16 | 14 |
| ① | ② | 3 | ④ | 5 | ⑥ | 12 | 9 | 15 | 7 | 10 | 13 | 8 | 11 | 16 | 14 |
| ① | ② | 3 | ④ | 5 | ⑥ | 8 | 9 | 11 | 7 | 10 | ⑫ | 13 | 15 | 16 | 14 |
| ① | ② | 3 | ④ | 5 | ⑥ | 7 | ⑧ | 11 | 9 | 10 | ⑫ | 13 | 15 | 16 | 14 |
| ① | ② | 3 | ④ | 5 | ⑥ | 7 | ⑧ | 10 | 9 | ⑪ | ⑫ | 13 | 15 | 16 | 14 |
| ① | ② | 3 | ④ | 5 | ⑥ | 7 | ⑧ | 9 | ⑩ | ⑪ | ⑫ | 13 | 15 | 16 | 14 |
| ① | ② | 3 | ④ | 5 | ⑥ | 7 | ⑧ | 9 | ⑩ | ⑪ | ⑫ | ⑬ | 15 | 16 | 14 |
| ① | ② | 3 | ④ | 5 | ⑥ | 7 | ⑧ | 9 | ⑩ | ⑪ | ⑫ | ⑬ | 14 | ⑮ | 16 |

Figure: An example of quicksort.
Source: redrawn from [Manber 1989, Figure 6.12].

**Average-Case Complexity of Quick Sort**

- When $X[i]$ is selected (at random) as the pivot,

$$T(n) = n - 1 + T(i-1) + T(n-i), \text{ where } n \geq 2.$$

The average running time will then be

$$\begin{aligned} T(n) &= n - 1 + \tfrac{1}{n}\sum_{i=1}^{n}(T(i-1) + T(n-i)) \\ &= n - 1 + \tfrac{1}{n}\sum_{i=1}^{n} T(i-1) + \tfrac{1}{n}\sum_{i=1}^{n} T(n-i) \\ &= n - 1 + \tfrac{1}{n}\sum_{j=0}^{n-1} T(j) + \tfrac{1}{n}\sum_{j=0}^{n-1} T(j) \\ &= n - 1 + \tfrac{2}{n}\sum_{i=0}^{n-1} T(i) \end{aligned}$$

- Solving this recurrence relation with full history, $T(n) = O(n \log n)$.

## 3.5 Heap Sort

**Heap Sort**

**Algorithm Heapsort** $(A, n)$;
**begin**
    $Build\_Heap(A)$;
    **for** $i := n$ **downto** 2 **do**
        $swap(A[1], A[i])$;
        $Rearrange\_Heap(i-1)$
**end**

Time complexity: $O(n \log n)$

**Heap Sort (cont.)**

**procedure Rearrange_Heap** $(k)$;
**begin**
    $parent := 1$;
    $child := 2$;
    **while** $child \leq k - 1$ **do**
        **if** $A[child] < A[child + 1]$ **then**
            $child := child + 1$;
        **if** $A[child] > A[parent]$ **then**
            $swap(A[parent], A[child])$;
            $parent := child$;
            $child := 2 * child$
        **else** $child := k$
**end**

**Heap Sort (cont.)**

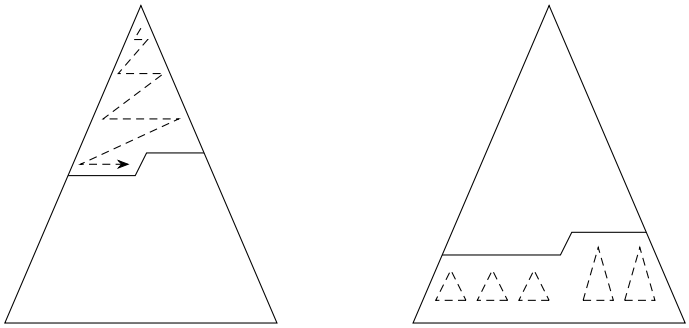


Figure: Top down and bottom up heap construction.
Source: redrawn from [Manber 1989, Figure 6.14].

How do the two approaches compare?

/* Top down: $O(n \log n)$.

Bottom up: $O(\text{sum of the heights of all nodes}) = O(n)$. Consider a full binary tree of height $h$. From an excercise problem in HW#2, we know that "sum of the heights of all nodes" of the tree equals $2^{h+1}-(h+2) \leq 2^{h+1}-1 = n$. */

### Building a Heap Bottom Up

| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 6 | 2 | 8 | 5 | 10 | 9 | 12 | 1 | 15 | 7 | 3 | 13 | 4 | 11 | 16 | 14 |
| 6 | 2 | 8 | 5 | 10 | 9 | 12 | (14) | 15 | 7 | 3 | 13 | 4 | 11 | 16 | (1) |
| 6 | 2 | 8 | 5 | 10 | 9 | (16) | 14 | 15 | 7 | 3 | 13 | 4 | 11 | (12) | 1 |
| 6 | 2 | 8 | 5 | 10 | (13) | 16 | 14 | 15 | 7 | 3 | (9) | 4 | 11 | 12 | 1 |
| 6 | 2 | 8 | 5 | 10 | 13 | 16 | 14 | 15 | 7 | 3 | 9 | 4 | 11 | 12 | 1 |
| 6 | 2 | 8 | (15) | 10 | 13 | 16 | 14 | (5) | 7 | 3 | 9 | 4 | 11 | 12 | 1 |
| 6 | 2 | (16) | 15 | 10 | 13 | (12) | 14 | 5 | 7 | 3 | 9 | 4 | 11 | (8) | 1 |
| 6 | (15) | 16 | (14) | 10 | 13 | 12 | (2) | 5 | 7 | 3 | 9 | 4 | 11 | 8 | 1 |
| (16) | 15 | (13) | 14 | 10 | (9) | 12 | 2 | 5 | 7 | 3 | (6) | 4 | 11 | 8 | 1 |

Figure: An example of building a heap bottom up.
Source: adapted from [Manber 1989, Figure 6.15].

### A Lower Bound for Sorting

- A lower bound for a particular problem is a proof that *no algorithm* can solve the problem better.
- We typically define a computation model and consider only those algorithms that fit in the model.
- **Decision trees** model computations performed by *comparison-based* algorithms.

**Theorem 6** (Theorem 6.1)**.** *Every decision-tree algorithm for sorting has height* $\Omega(n \log n)$.

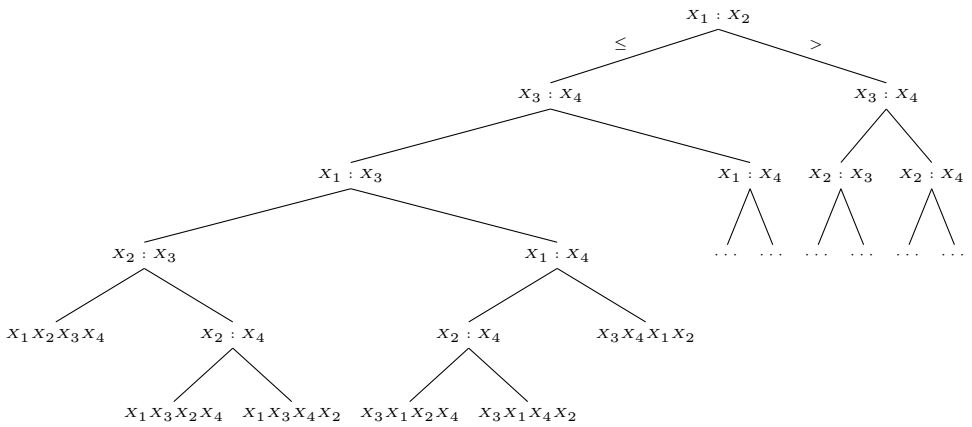
Proof idea: there must be at least $n!$ leaves in the decision tree, one for each possible outcome.

/* Recall Stirling's approximation: $n! = \sqrt{2\pi n}\left(\frac{n}{e}\right)^n (1 + O(1/n))$. The height of the decision tree must be at least $\log(n!)$, i.e., $\Omega(n \log n)$. */

Is the lower bound contradictory to the time complexity of radix sort?

**A Lower Bound for Sorting (cont.)**

A decision tree (partly shown) for the merge sort with $X_1X_2X_3X_4$ as input:




Note: in total, there should be $4! = 24$ leaves, only six of which are shown.

# 4 Order Statistics

**Order Statistics: Minimum and Maximum**

**Problem 7.** *Find the maximum and minimum elements in a given sequence.*

- The obvious solution requires $(n-1) + (n-2)$ $(= 2n-3)$ comparisons between elements.
- Can we do better? (Which comparisons could have been avoided?)

  /* A better algorithm: compare $x_1$ and $x_2$. Set $min$ to be the smaller of the two and $max$ the larger. Compare $x_3$ and $x_4$ and then compare the smaller with $min$ and the larger with $max$; these take three comparisons. Update $min$ and $max$ accordingly. Continue until we have exhausted the sequence of numbers. Assuming $n$ is even, the total number of comparisons $= 1 + 3 \times \frac{(n-2)}{2} = \frac{3}{2}n - 2$. */

**Order Statistics: $K$th-Smallest**

**Problem 8.** *Given a sequence $S = x_1, x_2, \cdots, x_n$ of elements, and an integer $k$ such that $1 \leq k \leq n$, find the $k$th-smallest element in $S$.*

**Order Statistics: $K$th-Smallest (cont.)**

```
procedure Select (Left, Right, k);
begin
    if Left = Right then
       Select := Left
    else Partition(X, Left, Right);
         let Middle be the output of Partition;
         if Middle − Left + 1 ≥ k then
            Select(Left, Middle, k)
         else
            Select(Middle + 1, Right, k − (Middle − Left + 1))
end
```

```
Algorithm Selection (X, n, k);
begin
    if (k < 1) or (k > n) then print "error"
    else S := Select(1, n, k)
end
```

/* Here the formal parameter $k$ (for rank) is made to be relative to the left bound of array indices, while $Left$, $Middle$, and $Right$ are absolute index values. */

**Order Statistics: $K$th-Smallest (cont.)**

The nested "**if**" statement may be simplified:

```
procedure Select (Left, Right, k);
begin
    if Left = Right then
       Select := Left
    else Partition(X, Left, Right);
         let Middle be the output of Partition;
         if Middle ≥ k then
            Select(Left, Middle, k)
         else
            Select(Middle + 1, Right, k)
end
```

# 5 Finding a Majority

**Finding a Majority**

**Problem 9.** *Given a sequence of numbers, find the majority in the sequence or determine that none exists.*

A number is a *majority* in a sequence if it occurs more than $\frac{n}{2}$ times in the sequence.

Caution: maintaining a counter for each possible number requires $O(\log n)$ time for each access to a particular counter, which means $O(n \log n)$ time in total. Sorting the sequence to find a probable candidate also requires $O(n \log n)$ time.

Idea: compare any two numbers in the sequence. What can we conclude if they are not equal?

/* If there is a majority, it is also a majority of the other $n - 2$ numbers. However, the reverse may not be true. */

What if they are equal?

/* Keep the first number as a candidate at hand and repeat the following:
If the next number equals the candidate, we increment the count of its occurrences; otherwise, we have a pair of unequal numbers to eliminate (by decrementing the count for the candidate). When the count becomes 0 (due to elimination), we take the next number as a new candidate. */

**Finding a Majority (cont.)**

```
Algorithm Majority (X, n);
begin
    C := X[1];  M := 1;
    for i := 2 to n do
```

```
        if M = 0 then
            C := X[i];  M := 1
        else
            if C = X[i] then M := M + 1
            else M := M − 1;
```

**Finding a Majority (cont.)**

```
    if M = 0 then Majority := −1
    else
        Count := 0;
        for i := 1 to n do
            if X[i] = C then Count := Count + 1;
        if Count > n/2 then Majority := C
        else Majority := −1
end
```

Time complexity: $O(n)$.