Lower Bound for Sorting Complexity

- Each algorithm that sorts by comparing only pairs of elements must use at least

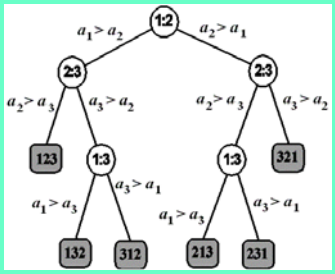
$$\lceil \log_2(n!) \rceil \cong n \log_2 n - 1.44n$$

comparisons in the worst case (that is, for some "worst" input sequence) and in the average case.

- Stirling's approximation of the factorial ($n!$):

$$1 \cdot 2 \cdot \ldots \cdot n \equiv n! \geq \left(n/e\right)^n \sqrt{2\pi n} \approx 2.5 n^{n+0.5} e^{-n}$$

Decision Tree for Sorting *n* Items



Decision tree for n =3:

- *i:j* - a comparison of a_i and a_j
- *ijk* - a sorted array $(a_i a_j a_k)$
- $n!$ permutations → $n!$ leaves

Sorting in descending order of the numbers

Decision Tree for Sorting *n* Items

- Decision tree for n=3: an array $\boldsymbol{a} = \{a_1, a_2, a_3\}$
- Example: {35, 10, 17}
  - Comparison 1:2 (35 > 10) → left branch $a_1 > a_2$
  - Comparison 2:3 (10 < 17) → right branch $a_2 < a_3$
  - Comparison 1:3 (35 > 17) → left branch $a_1 > a_3$
- Sorted array 132 → $\{a_1=35, a_3=17, a_2=10\}$

Decision Tree

- Decision tree of height h has $L_h \leq 2^h$ leaves
- Mathematical induction:
  - $h = 1$: the tree of height 1 has $L_1 \leq 2^h$ leaves
  - $h-1 \rightarrow h$: let the tree of height $h-1$ have $L_{h-1} \leq 2^{h-1}$ leaves; the tree of height h consists of a root and two subtrees at most of height $h-1$. Thus,

$$L_h = L_{h-1} + L_{h-1} \leq 2^{h-1} + 2^{h-1} = 2^h$$

Worst-Case Complexity of Sorting

- The lower bound for the least height h of a decision tree for sorting by pairwise comparisons which provides $L_h = 2^h \geq n!$ leaves is

$$h \geq \log_2(n!) \cong n \log_2 n - 1.44\, n$$

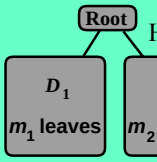
- Thus, the worst-case complexity of the sorting is at least $O(n \log n)$

Average-Case Sorting Complexity

- Each n-element decision tree has an average height at least $\log(n!) \geq n \log n$
- Let $H(D,k)$ be the sum of heights for all k leaves of a tree D and $H(k) = \min_D H(D,k)$ denote the minimum sum of heights
- Math induction to prove that $H(k) \geq k \log k$
  - $k = 1$: Obviously, $H(1) = 0$
  - $k-1 \rightarrow k$: Let $H(m) \geq m \log m$, $m < k$

Average-Case Sorting Complexity

- The tree D with k leaves contains 2 subtrees, D_1 with $m_1 < k$ leaves and D_2 with $m_2 < k$ leaves just under the root ($m_1 + m_2 = k$):



$$H(D,k) = k + H(D_1, m_1) + H(D_2, m_2)$$

because the link to the root adds 1 to each leaf's height

Average-Case Sorting Complexity

The minimum height : $H(k) = k + \min_{m_1+m_2=k} \{H(m_1) + H(m_2)\}$

By the induction assumption :

$$H(k) \le k + \min_{m_1+m_2=k} \{m_1 \log m_1 + m_2 \log m_2\}$$

The minimum is reached by symmetry for $m_1 = m_2 = \frac{k}{2}$:

$$H(k) \le k + 2\left(\tfrac{k}{2}\right)\log\left(\tfrac{k}{2}\right) = k \log k$$

Thus the average height of the decision tree with at least $n!$ leaves is : $\left(H(n!)/n!\right) \le \log_2 n! \cong n \log_2 n - 1.44n$

Data Search

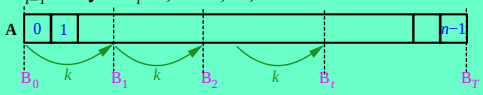
- Data record → Specific **key**
- Goal: to find all records with **key**s matching a given search **key**
- Purpose:
  - to access information in the record for processing, or
  - to update information in the record, or
  - to insert a new record or to delete the record

Types of Search

- **Static search**: stored data is not changed
  - Given an integer search key X, return either the position of X in an array A of records or an indication that it is not present without altering the array A
  - If X occurs more than once, return any occurrence
- **Dynamic search**: the data may be inserted or deleted

Sequential and Jump Search

- Sequential search is the only one for an **unsorted** array
- Successful / unsuccessful search: the $O(n)$ worst-case and average-case complexity
- Jump search $O(n^{0.5})$ in a **sorted** array **A** of size n: $T = \lceil n/k \rceil$ jumps of length k to the points $B_t = \min\{t \cdot k, n\}$ and the sequential search among k items in a t-th part such that $B_{t-1} \le$ **key** $\le B_t - 1$; $t = 1,\ldots,T$



Jump Search $O(n^{0.5})$

Worst-case complexity:

$$T(n,k) = \frac{n}{k} + k$$

Minimum complexity : for $k = \sqrt{n}$

$$T(n) = \min_k \{T(n,k)\} = 2\sqrt{n}$$

because $\frac{d}{dk} T(n,k) = -\frac{n}{k^2} + 1 = 0 \;\rightarrow\; n = k^2$

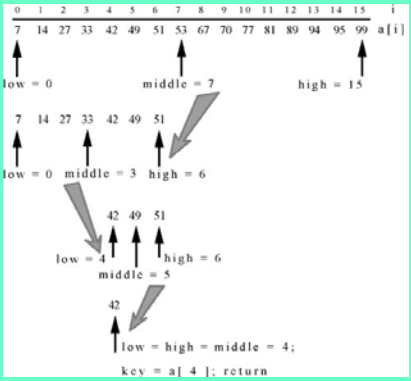
Binary Search O(log *n*)

- Ordered array: $\mathbf{key}_0 < \mathbf{key}_1 < \ldots < \mathbf{key}_{n-1}$
- Compare the search **key** with the record $\mathbf{key}_i$ at the middle position $i = \lfloor (n-1)/2 \rfloor$
  - if $\mathbf{key} = \mathbf{key}_i$, return i
  - if $\mathbf{key} < \mathbf{key}_i$ or $\mathbf{key} > \mathbf{key}_i$, then it must be in the 1st or in the 2nd half of the array, respectively
- Apply the previous two steps to the chosen half of the array iteratively (repeating halving principle)

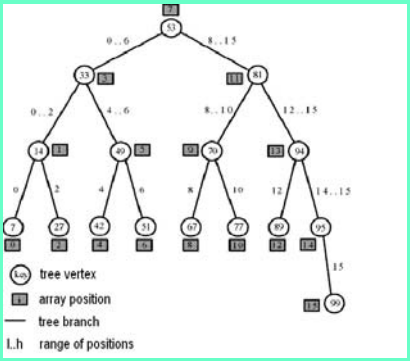
Implementation of Binary Search

```
begin BinarySearch (an integer array a of size n, a search key)
    low ← 0;   high ← n – 1
    while low ≤ high do
        middle ← ⌊( low + high ) / 2⌋
        if a[ middle ] < key  then  low  ←  middle + 1
        else if a[ middle ] > key then  high  ←  middle – 1
        else return  middle end if
    end while
    return ItemNotFound
end BinarySearch
```


Binary search: detailed analysis



Comparison structure: the binary (search) tree



Worst-Case Complexity O(log *n*) of Binary Search

- Let $n = 2^k - 1$; $k = 1,2,\ldots$, then the binary tree is complete (each internal node has 2 children)
- The tree height is $k - 1$
- Each tree level l contains 2^l nodes for $l = 0$ (the root), $1, \ldots, k-2, k-1$ (the leaves)
- $l + 1$ comparisons to find a key of level l
- The worst case: $k = \log_2(n+1)$ comparisons

Average-Case Complexity O(log *n*) of Binary Search

- Let $C_i = l + 1$ be the number of comparisons to find $\mathbf{key}_i$ of level l; $i = 0, \ldots, n-1$; $l = 0, \ldots, k-1$
- Average number: $\overline{C} = \frac{1}{n}\left(C_0 + C_1 + \ldots + C_{n-1}\right)$
- There are 2^l nodes at the level l, so that:

$$C_0 + C_1 + \ldots + C_{n-1} \equiv S_{k-1} = 1 \cdot 2^0 + \ldots + k \cdot 2^{k-1}$$

- By math induction: $S_{k-1} = 1 + (k-1)\,2^k$, so that

$$\overline{C} = \frac{1}{n}\left(1 + (k-1)2^k\right) = \frac{n+1}{n}\log_2(n+1) - 1$$