Time Complexity

	Notation	Meaning	Equivalent Symbol
Big-Oh	O	Upper bounds	$\leq$
Little-oh	o	Strict upper bound	$<$
Big-Omega	Ω	Lower bounds	$\geq$
Little-omega	ω	Strict lower bound	$>$
Big-Theta	Θ	Tight bounds	$=$

Tight bounds. T(n) is Θ(f(n)) if there exist constants $c_1 > 0$, $c_2 > 0$ and $n_0 \geq 0$ such that $c_1 \cdot f(n) \leq T(n) \leq c_2 \cdot f(n)$ for all $n \geq n_0$.

Substitution method

replace the T(n) with c·f(n)

Master Method

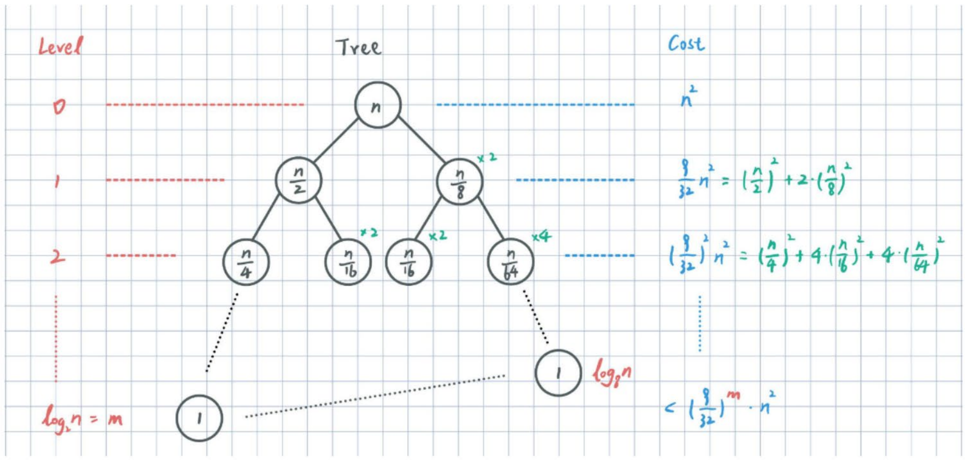
- "Cookbook" for solving recurrences of the form:

$$T(n) = aT\left(\frac{n}{b}\right) + f(n)$$

where, a ≥ 1, b > 1, and f(n) > 0

- **Case 1:** if $f(n) = O(n^{\log_b a - \epsilon})$ for some $\epsilon > 0$, then $T(n) = \Theta(n^{\log_b a})$;
- **Case 2:** if $f(n) = \Theta(n^{\log_b a})$, then $T(n) = \Theta(n^{\log_b a} \log n)$;
- **Case 3:** if $f(n) = \Omega(n^{\log_b a + \epsilon})$ for some $\epsilon > 0$, and if $af(n/b) \leq cf(n)$ for some $c < 1$ and all sufficiently large n, then $T(n) = \Theta(f(n))$.

Iteration Method (Tree)



Divide and Conquer

maximum-subarray: T(n)=2T(n/2)+Θ(n), T(n)=Θ(nlogn)

Useful conclusions

maximum-subarray: dividing < combining

merge-sort: dividing < combining

quick-sort: dividing > combining

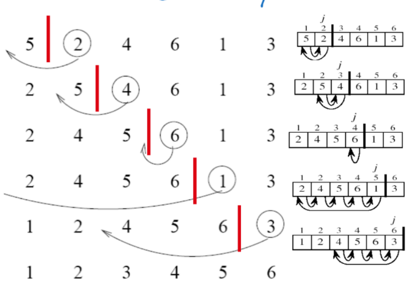
Asymptotic analysis

- Ignoring the constant factor
  - 347n is Θ(n)
- Constant factors of exponentials cannot be ignored
  - 2^{3n} is not $O(2^n)$
- Concentrating on trends of the large value of n
  - $3n^2 \log n + 25n \log n$ is $\Theta(n^2 \log n)$

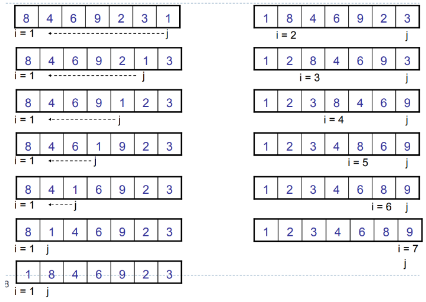
Sorting

Insertion sort

"almost sorted" arrays $\Theta(n)$



Bubble sort



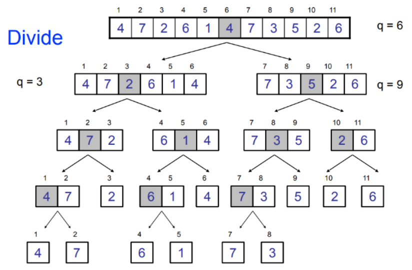
Loop Invariants

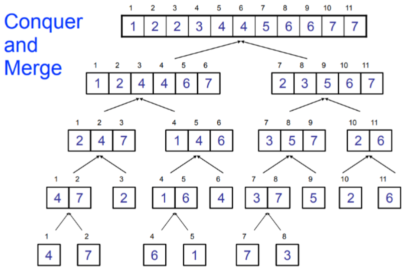
Proving loop invariants works like induction

- Initialization (base case):
  - It is true prior to the first iteration of the loop
- Maintenance (inductive step):
  - If it is true before an iteration of the loop,
    it remains true before the next iteration
- Termination:
  - When the loop terminates, the invariant gives us
    a useful property that shows the algorithm is correct
  - Stop the induction when the loop terminates

Merge sort

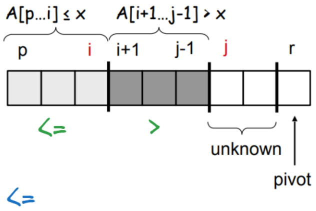
Guaranteed to run in $\Theta(n\lg n)$, but requires extra space $\approx n$





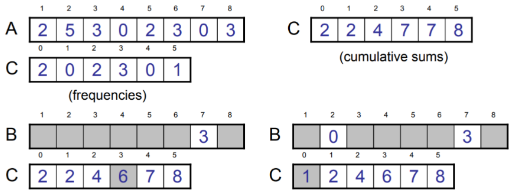

```python
def QuickSort(A, p, r):
    if p < r:
        q = Partition(A, p, r)
        QuickSort(A, p, q)
        QuickSort(A, q+1, r)
```

Lomuto partition



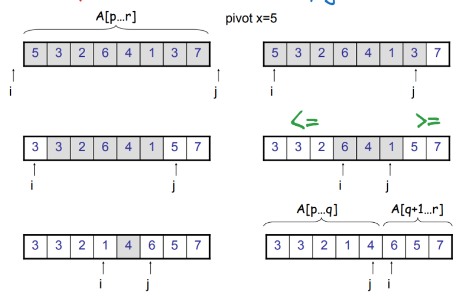
Count sort

use when $r = O(n)$, then $T(n) = \Theta(n)$



Hoare partition

i until $>=$, j until $<=$



Radix sort

Example: key=15

- $key_{10} = 15$, d=2, k=10 where $0 \le x_i \le 9$
- $key_2 = 1111$, d=4, k=2 where $0 \le x_i \le 1$

when use count to sort each digit, $T(n) = \Theta(d(n+k))$

Selection Problem

find min and max in paris, 3 comparison per two elements, $O(3n/2)$

Randomized Select

use Randomized Partition,

best case: $T(n) = T(\alpha \cdot n) + O(n)$, where $\alpha < 1$, $T(n) = O(n)$

worst case: $T(n) = T(n-1) + O(n)$, $T(n) = O(n\char`^2)$

Worst-Case Linear-Time Selection

- step1: divide n elements into n/5 groups
- step2: find median of each group of 5 elements, using insertion-sorting
- step3: recursively SELECT to find the median x of n/5 medians found in step 2
- step 4: use x found in step 3 to partition the array, let k be one plus the number of elements in the low side of partition
- step 5: if i=k, return x; else recursively SELECT i-th element in low side, if i<k; or SELECT (i-k)-th element in high side, if i>k.

Useful conclusions

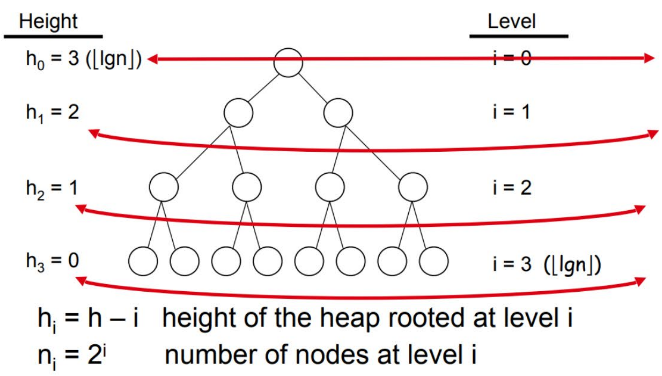
If use lomuto partition, the probability that the i-th ranked element will be compared against (i+k)-th element is 2/(k+1)

Algorithm	Time complexity (average case)	Best case	Worst case	In-place	Stable
Insertion sort	$O(n^2)$	$O(n)$	$O(n^2)$	Yes	Yes
Bubble sort	$O(n^2)$	$O(n)$	$O(n^2)$	Yes	Yes
Merge sort	$O(n \log n)$	$O(n \log n)$	$O(n \log n)$	No	Yes
Heap sort	$O(n \log n)$	$O(n \log n)$	$O(n \log n)$	Yes	No
Quick sort	$O(n \log n)$	$O(n \log n)$	$O(n^2)$	Yes	No
Count sort	$O(n + k)$	$O(n + k)$	$O(n + k)$	No	Yes
Radix sort	$O(nk)$	$O(nk)$	$O(nk)$	No	Yes

Heap

Item	Notation
Node i	$A[i]$
Root of the tree	$A[1]$
Left child	$A[2i]$
Right child	$A[2i+1]$
Parent	$A[\lfloor i/2 \rfloor]$

Operation	Running time
Max-Heapify	$O(\log n)$
Build-Max-Heap	$O(n)$
Heap-Sort	$O(n \log n)$
Extract-Max	$O(\log n)$
Increase-Key	$O(\log n)$
Max-Heap-Insert	$O(\log n)$



operations

Max-Heapify: Recursively exchange the node with its larger child

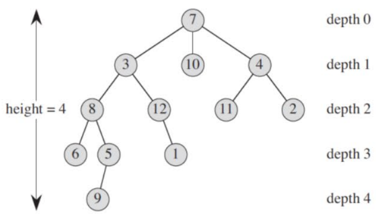
Build-Max-Heap: call Max-Heapify on A[n/2], ..., A[1]

Heap-Sort: swap A[n] and A[1], call Max-Heapify on A[1]

Increase-Key: recursively swap the newly inserted node with its parent, (if larger than its parent)

Tree

- **degree** of x: number of children
- **depth** of x: length of the simple path from root to x
- **level** of a tree: all nodes at the same depth
- **height** of x: length of the longest simple path from x downward to some leaf node
- **height of a tree**: height of root



Hash table

Load factor(expected list length): $\alpha=n/m$

Universal hashing function: h(k) and h(l) are independent

Hashing methods

Division method: h(k) = k mod m

m choose a prime, not close to power of 2

Multiplication Method: $h(k) = \lfloor m(kA - \lfloor kA \rfloor) \rfloor$

$0 < A < 1$, m typically power of 2

Binary Search Tree

Traversal

pre-order: root-left-right

in-order: left-root-right

in-order: left-right-root

successor of x: the smallest key > x

predecessor of x: the biggest key < x

Operations all require O(h)

Useful conclusions

1. expected number of elements exanmined in an unseccessful search is α
2. The expected number of elements examined when searching for the k-th inserted key is 1+(n-k)/m
3. The expected number of elements examined when searching for a randomly key is 1+(n-1)/2m

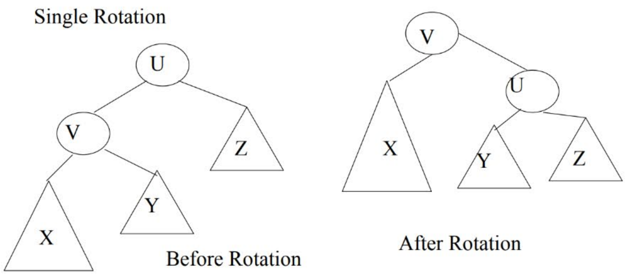
$$1+\frac{1}{n}\sum_{k=1}^{n}\frac{n-k}{m}=\frac{n-1}{2m}+1$$

$$\begin{aligned}
\mathrm{E}\left[\frac{1}{n}\sum_{i=1}^{n}\left(1+\sum_{j=i+1}^{n}X_{ij}\right)\right] &= \frac{1}{n}\sum_{i=1}^{n}\left(1+\sum_{j=i+1}^{n}\mathrm{E}[X_{ij}]\right) \\
&= 1+\frac{n-1}{2m} \\
&= 1+\frac{\alpha}{2}-\frac{\alpha}{2n}.
\end{aligned}$$

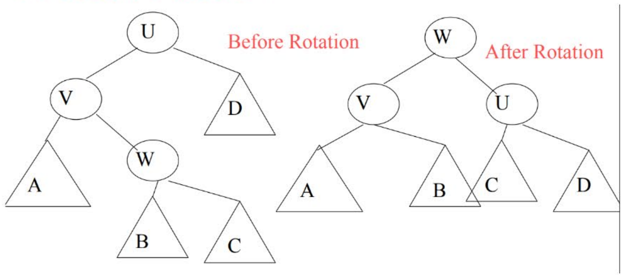
AVL tree

height of the two subtrees of a node differs by at most one

single rotation



double rotation



Shrink and paste!

(c) **T or F:** By using uniform hashing functions, the number of elements hashed into a slot is independent from the number of elements in any other slot;

(d) **T or F:** Bubble sort is stable;

(b) **T or F:** Radix sort follows the divide-and-conquer design;

(c) **T or F:** The worst-case running time and average-case running time are equal asymptotically (same order) for any randomized algorithms;

(d) **T or F:** Quicksort with Hoare's partition is stable;

$$P(\{\max_{i=1}^{m} X_i = k\}) \le P(\cup_{i=1}^{m}\{X_i = k\}) \le \sum_{i=1}^{m} P(X_i = k) = mP(X_i = k);$$