Orders of Growth

Recurrence Relation	Time Complexity	Remarks
$T(n) = \theta(1) + T(n-1)$	$\theta(n)$	build_list(f, n), append, linear search
$T(n) = \theta(n) + T(n-1)$	$\theta(n^2)$	selection sort, insertion sort
$T(n) = \theta(1) + 2T(\frac{n}{2})$	$\theta(\log n)$	Binary search, tree traversal
$T(n) = \theta(n) + 2T(\frac{n}{2})$	$\theta(n \log n)$	Merge sort
$T(n) = \theta(1) + 2T(n-1)$	$\theta(2^n)$	Tree recursive fibonacci
$T(n) = \theta(1) + T(\sqrt{n})$	$\theta(log(logn))$	N.A
$T(n) = \theta(\sqrt{n}) + \sqrt{n}T(\sqrt{n})$	$\theta(nlog(logn))$	Let $n = 2^k$ and Master's Theorem

In general, $T(n) = \theta(n^k) + T(n-1) \rightarrow T(n) = \theta(n^{k+1})$

Master Theorem

If T(n)= $aT(\frac{n}{b}) + f(n)$, $f(n) = n^k$, then

$$T(n) = \begin{cases} n^k, & a < b^k \\ n^k \log_b n, & a = b^k \\ n^{\log_b a}, & a > b^k \end{cases}$$

If $a_n = a_{n-1} + c$

$$\sum_{i=1}^{n} a_i = a_1 + a_2 + a_3 + \cdots + a_n$$

$$= \frac{n(a_n + a_1)}{2}$$

$$\sum_{i=1}^{n} ar^{i-1} = a + ar + ar^2 + \cdots + ar^{n-1}$$

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

If $0 < r < 1$

$$\sum_{i=1}^{\infty} ar^{i-1} = \frac{a_1}{1 - r}$$

$$\sum_{i=1}^{n} \frac{1}{i} = 1 + \frac{1}{2} + \frac{1}{3} + \cdots + \frac{1}{n}$$

$$\approx ln(n+1)$$

$$\sum_{i=1}^{n} i^2 = 1 + 4 + 9 + \cdots + n^2$$

$$= \frac{n(n+1)(2n+1)}{6}$$

$$log_b a = \frac{1}{log_a b}$$

$$log_a x = \frac{log_b x}{log_b a}$$

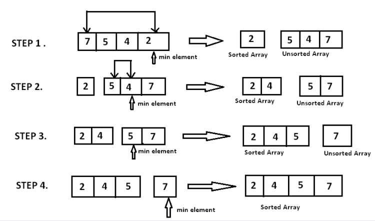
$$log_2(n!) = n \cdot log_2 n$$

Function	Name
5 O(1)	Constant
loglog(n)	double log
log(n)	logarithmic
$\log^2(n)$	Polylogarithmic
n	linear
nlog(n)	log-linear
n^3	polynomial
$n^3\log(n)$	
n^4	polynomial
2^n	exponential
2^{2n}	
n!	factorial

Sorting

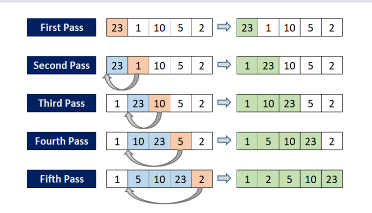
Selection Sort

```
for j in [1:len(A)]:
    k = indexOfMin(A[j..len(A)])
    swap(A[j], A[k])
```



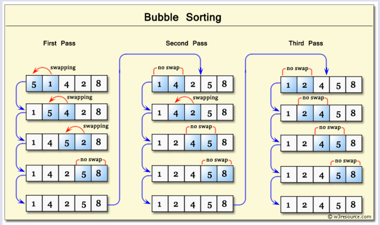
Insertion Sort

```
for i in [1:len(A)]:
    key = A[i]
    j = i - 1
    # find correct position for key within
    A[1:j]
    while (j >= 0) and (A[j] > key):
        # move element to the right ('make
        space for key')
        A[j+1] = A[j]
        j -= 1
    # insert key here
    A[j+1] = key
```



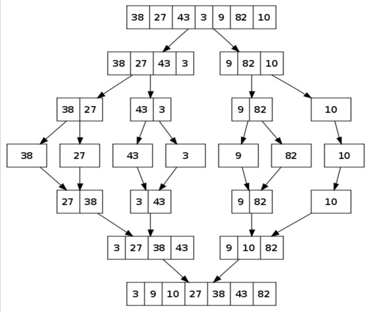
Bubble Sort

```
repeat (until no swaps):
    for j in [0 : len(A)-1]:
        if (A[j] > A[j+1]):
            swap(A[j], A[j+1])
```

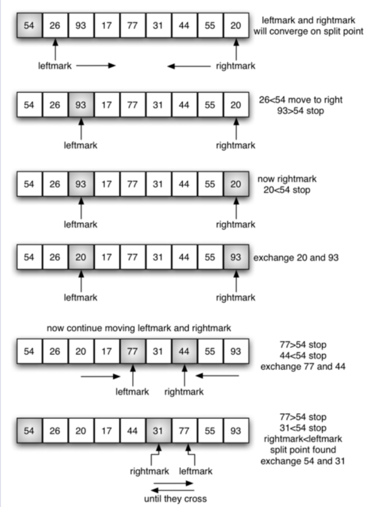


Merge Sort

```
if (len(A) <= 1):
    return
else:
    mid = len(A) // 2
    left = mergeSort(A[0:mid])
    right = mergeSort(A[mid:len(A)])
    return merge(left, right)
```



QuickSort



Properties

Algorithm	*Stability*	*In-place*	*Invariant*
Selection	✗	✓	At the end of iteration j: the j smallest items in the array are sorted.
Insertion	✓	✓	At the end of iteration j: the first j items in the array are in sorted order.
Bubble	✓	✓	At the end of iteration j: the j largest items in the array are sorted.
Merge	✓	✗	N.A
Quick	✗	✓	1. Pivot is in correct position at the end of partitioning. 2. For all `1 < i < low, A[i] < pivot` 3. For all `j >= high, A[j] > pivot`

Time Complexity

Algorithm	*Unsorted*	*Sorted*	*Reverse Sorted*	*Almost Sorted*
Selection	$O(n^2)$	$O(n^2)$	$O(n^2)$	$O(n^2)$
Insertion	$O(n^2)$	$O(n)$	$O(n^2)$	$O(n)$
Bubble	$O(n^2)$	$O(n)$	$O(n^2)$	$O(n^2)$
Merge	$O(nlogn)$	$O(nlogn)$	$O(nlogn)$	$O(nlogn)$
Quick	$O(nlogn)$	$O(n^2)$	$O(n^2)$	$O(nlogn)$

Swaps

Algorithm	*Best Case*	*Worst Case*
Selection	0 (Sorted)	$O(n)$
Insertion	0 (Sorted)	$O(n^2)$ (Reverse)
Bubble	0 (Sorted)	$O(n^2)$ (Reverse)
Merge	0	0 (Only Copying)
Quick	$O(nlogn)$	$O(n^2)$ (Reverse)

Comparisons

Algorithm	*Best Case*	*Worst Case*
Selection	$O(n^2)$	$O(n^2)$
Insertion	$0(n^2)$ (Sorted)	$O(n^2)$
Bubble	$0(n^2)$ (No Flag)	$O(n^2)$
Merge	$O(nlogn)$	$O(nlogn)$
Quick	$O(nlogn)$	$O(n^2)$

Space Complexity

Algorithm	*Extra Memory*
Selection	$O(1)$
Insertion	$O(1)$
Bubble	$O(1)$
Merge	$O(nlogn)$
Quick	$O(n)$ (average: $O(logn)$)

Binary Search

```
def search(A,  key,  begin,  end):
    if (begin > end): return -1
    # avoid integer overflow errors
    mid = begin + (end-begin)/2
    if (key < A[mid]):
        # eliminate right half
        return search(A, key, begin, mid)
    else if (key > A[mid]):
        # eliminate left half
        return search(A, key, mid+1, end)
    else: return mid
```

1. Given a function `complicatedFunction(input)` that is **monotonic increasing**
2. i.e. `complicatedFunction(i) < complicatedFunction(i+1)`
3. Task: Find the minimum value `j` such that: `complicatedFunction(j) > num`

```
def bisectRight(A, key):
    # returns the index of the first value
    strictly greater than key
    low = 0, high = len(A) - 1
    if (A[high] < key): return -1
    while (low < high)
        mid = (low + high)/2
        if (A[mid] <= key): low = mid+1
        else if (A[mid] > key): high = mid
    return low
```

AVL Trees

1. Weight: Number of nodes in the subtree rooted at the node.

(a) `weight(null) = 0`

(b) `weight(leaf) = 1`

(c) `weight(u) = w.left.weight + w.right.weight + 1`

2. Number of edges on the path from the node to the deepest leaf.

(a) `height(empty tree) = -1`

(b) `height(u) = max(u.left.height, u.right.height) + 1)`

- BST is balanced if $h = O(logn)$, i.e. $c \cdot logn$, allowing all operations to run in $O(logn)$ time
- A node `u` is said to be height-balanced if $|$`u.left.height - u.right.height`$| \leq 1$.
- BST is height-balanced if every node is height-balanced

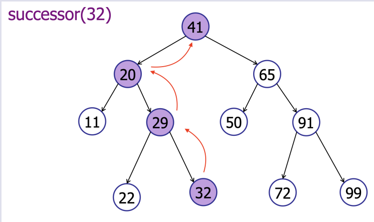
Notes

- Height-balanced ⇒ Balanced
- Balanced ⇏ Height-balanced
- A height-balanced tree has height $O(2logn) = O(logn)$
- Define the balance factor of a node u as `balance(u) = u.left.height - u.right.height`. When $|$`balance(u)`$| \geq 2$, rebalancing is required. This must be done from the insertion/deletion point **up to the root**.

Successor

Successor

```
if (u.right != null):
    return findMin(u.right)
else:
    p = u.parent
    # find an ancestor that is a left child
    while (p is a right child):
        go up the ancestry
    if (p == null): return null
    else: return parent
```



Case 2: node has no right child.

Rank & Select

The rank of an element is its position relative to the sorted order, i.e. the kth smallest item would have a rank of k. Select is the reverse of rank. Given a rank, return the value of the node with that rank.

Rank & Select

```
# computes the rank of `u` within the
subtree rooted at `root`
getRank(u, root):
    if (u.key < root.key):
        return getRank(u, root.left)
    else if (u.key > root.key):
        return root.left.weight + 1 +
        getRank(u, root.right)
    else:
        return root.left.weight + 1

select(rank, root):
    # this is equivalent to calling
    getRank(root, root)
    rankOfRootInSubTree = root.left.weight +
    1
    if (rank < rankOfRootInSubTree):
        return select(rank, root.left)
    else if (rank > rankOfRootInSubTree):
        // eliminate the root and its right
        subtree
        return select(rank -
        rankOfRootInSubTree, root.right)
    else:
        return root
```

Rotations

A node is **left-heavy** if its left sub-tree is **taller** than the right sub-tree, i.e. `node.left.height > node.right.height`.

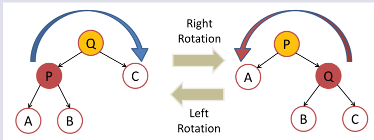
1. If v is out of balance and left heavy:

(a) `v.left` is **balanced**: `rightRotate(v)`

(b) `v.left` is **left-heavy**: `rightRotate(v)`

(c) `v.left` is **right-heavy**: `leftRotate(v.left)` **and** `rightRotate(v)`

2. If v is out of balance and right heavy: Symmetrical cases, e.g. if `v.right` is **balanced**: `rightRotate(v)`



Notes

- Right rotations require a left child, left rotations require a right child
- The maximum number of rotations required upon insertion is 2, while for deletion it is $O(logn)$

Maximum Value

Each node `u` is augmented with: `value`, that specifies the value associated with that node, and `max`, the maximum `value` for any node in the subtree rooted at `u`

New Operations
`updateValue(key, newValue)`

1. Search the tree in the usual way for the specified `key`
2. Assuming a node `u` was found, update `u.value = newValue`
3. Update the tree - for every node `v` on the path from `u` to `root`, update `v.max = max(v.left.max, v.right.max, v.value)`

Note: Make sure to check children for `null`
Maintenance

- When performing a rotation on `u`, only `u` and `u.parent` change. Let `v = u.parent`. After a rotation of `u`, set `u.max = v.max`, and `v.max = max(v.value, v.left.value, v.right.value)` (be sure to avoid `NullPointerException`)
- When a node `u` is inserted: Set `u.value = <initial>` and `u.max = <initial>`. Perform rotations to rebalance.
- When a node u is deleted:

1. if `u` is a leaf, we can just delete it. For every ancestor v of `u`, update `v.max = max(v.left.max, v.right.max, v.points)`
2. if `u` has one child, then delete `u`, connecting `u.parent` to `u.child`. For every node `v` on the path from `u` to `root`: `v.max = max(v.left.max, v.right.max, v.points)`
3. node `u` has two children. Let `v = successor(u)`. Delete v from the tree, and for every node `w` on the path from `v` to u, update `w.max = max(w.left.max, w.right.max, w.points)`. Then replace `u` with `v`, and continue to update every node `w` on the path from `v` to the `root`.

Perform rotations to rebalance.

Total Count

Each node `u` is augmented with: `value`, that specifies the value associated with that node, and `max`, the maximum `value` for any node in the subtree rooted at `u`

New Operations
`updateValue(key, newValue)`

1. Search the tree in the usual way for the specified `key`
2. Assuming a node `u` was found, update `u.value = newValue`
3. Update the tree - for every node `v` on the path from `u` to `root`, update `v.max = max(v.left.max, v.right.max, v.value)`
   Note: Make sure to check children for `null`

Maintenance Each node `u` is augmented with: `value`, that specifies the value associated with that node, and `total`, the count of the number of special nodes in the subtree rooted at `u`

New Operations
`addToValue(key, val)`

1. Search the tree in the usual way for the specified `key`
2. Assuming a node `u` was found, update `u.value += val`
3. If `isSpecial(u.value)`, update the tree - for every node `v` on the path from `u` to `root`, update `v.total += 1`

Note: The above is symmetrical for `subtractFromValue`

`searchSpecial()`

1. Let `v = root`
2. Base Case: if `isSpecial(v.value)`, return `v`
3. If `v.total == 0`, return `null`
4. Else if `v.isLeaf()`, return `v`
5. Else if `v.left.total > 0`, recurse on `v.left`
6. Else recurse on `v.right`

Note: Avoid `NullPointerException`

Maintenance
Similar to **Max Value** augmentation, except that a node `u` is updated as follows:
`u.total = u.left.total + u.right.total`

Cheatsheet Examples

Key concepts I

Bag-of-words (BOW)

type* vs. *token

To be or not to be. = 6 *tokens*, 4 *types*.

type

descriptive criterion[a]

token

unit of analysis[b]

Key topics

- One
- Two
- Three

[a]**stroustrup**.
[b]**stroustrup**; **attentionMerchants**.

1 Tools

Lorem Ipsum

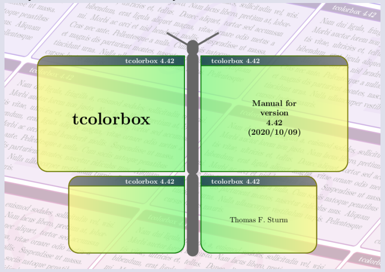
test ● test ● test ● test

Visualize

1. **present** your data
2. **analyze** the information
3. **explore** the findings

Voyant Tools

Voyant Tools ● Voyant Tools



Project Gutenberg Texts

84	Frankenstein; Or, The Modern Prometheus by Mary Wollstonecraft Shelley
6087	The Vampyre; a Tale by John William Polidori
696	The Castle of Otranto by Horace Walpole
42	The Strange Case of Dr. Jekyll and Mr. Hyde by Robert Louis Stevenson

Key concepts

Bag-of-words (BOW)

Zipf's Law

`_&Âğ!$Âğ/()$` .... code
`shutdown -h now` .... to shutdown

2 Programming

2.1 Code boxes

Code box using R

```r
# Install
install.packages("tm")  # for text mining

# Load
library("tm")

# text <- readLines(file.choose())
# Read the text file from internet
filePath <- "http://www.internet.com/text.txt"
text <- readLines(filePath)
```

Code box using C++

```cpp
for (auto element : vector)
{
    sum += element;
}
```

3 Graphics

The following is an example for a custom graphics command

4 Other types of boxes

The Alert Block

Suspendisse vitae elit. Aliquam arcu neque, ornare in, ullamcorper quis, commodo eu, libero. Fusce sagittis erat at erat tristique mollis. Maecenas sapien libero, molestie et, lobortis in, sodales eget, dui. Morbi ultrices rutrum lorem. Nam elementum ullamcorper leo. Morbi dui. Aliquam sagittis. Nunc placerat. Pellentesque tristique sodales est. Maecenas imperdiet lacinia velit. Cras non urna. Morbi eros pede, suscipit ac, varius vel, egestas non, eros. Praesent malesuada, diam id pretium elementum, eros sem dictum tortor, vel consectetuer odio sem sed wisi.

The Example Block

Etiam suscipit aliquam arcu. Aliquam sit amet est ac purus bibendum congue. Sed in eros. Morbi non orci. Pellentesque mattis lacinia elit. Fusce molestie velit in ligula. Nullam et orci vitae nibh vulputate auctor. Aliquam eget purus. Nulla auctor wisi sed ipsum. Morbi porttitor tellus ac enim. Fusce ornare. Proin ipsum enim, tincidunt in, ornare venenatis, molestie a, augue. Donec vel pede in lacus sagittis porta. Sed hendrerit ipsum quis nisl. Suspendisse quis massa ac nibh pretium cursus. Sed sodales. Nam eu neque quis pede dignissim ornare. Maecenas eu purus ac urna tincidunt congue.

`https://github.com` ...... Github Link Example

Branch

Explanation

Commit

What's a commit?

Staging

s. Index

Lipsum

Nunc velit. Nullam elit sapien, eleifend eu, commodo nec, semper sit amet, elit. Nulla lectus risus, condimentum ut, laoreet eget, viverra nec, odio. Proin lobortis. Curabitur dictum arcu vel wisi. Cras id nulla venenatis tortor congue ultrices. Pellentesque eget pede. Sed eleifend sagittis elit. Nam sed tellus sit amet lectus ullamcorper tristique. Mauris enim sem, tristique eu, accumsan at, scelerisque vulputate, neque. Quisque lacus. Donec et ipsum sit amet elit nonummy aliquet. Sed viverra nisl at sem. Nam diam. Mauris ut dolor. Curabitur ornare tortor cursus velit.

Yay Quotes

> “Yay, a quote!”

> “Yay, a longer quote! Fusce mauris. Vestibulum luctus nibh at lectus. Sed bibendum, nulla a faucibus semper, leo velit ultricies tellus, ac venenatis arcu wisi vel nisl. Vestibulum diam. Aliquam pellentesque, augue quis sagittis posuere, turpis lacus congue quam, in hendrerit risus eros eget felis. Maecenas eget erat in sapien mattis porttitor. Vestibulum porttitor. Nulla facilisi. Sed a turpis eu lacus commodo facilisis. Morbi fringilla, wisi in dignissim interdum, justo lectus sagittis dui, et vehicula libero dui cursus dui. Mauris tempor ligula sed lacus. Duis cursus enim ut augue. Cras ac magna. Cras nulla. Nulla egestas. Curabitur a leo. Quisque egestas wisi eget nunc. Nam feugiat lacus vel est. Curabitur consectetuer.”