I. Logarithm Rules

Inverse: $2^{\log_2 n} = \log_2(2^n) = n$

Change of Base: $\log_a n = \dfrac{\log_b n}{\log_b a}$

II. Recurrence Relation

$\underbrace{T(n)}_{O(\phi^n)} = T(n-1) + T(n-2) + 1$

$O(\phi^n)$; $\phi = \dfrac{1+\sqrt{5}}{2} \approx 1.618$

Fibonacci Sequence

III. Tips for Algorithms

- $(a_1, b_1), (a_2, b_2), \dots, (a_n, b_n)$ ⇒ Sort goal: by a then b

Solution: Sort by b_i first → Sort by a_i < with stable algorithm >

Note: Can use unstable algorithm for b_i

- Check feasibility of Randomized Algo. ⇒ Every outputs have equal probabilities!

IV. Proof of Correctness ⇒ Show that $\boxed{P(i)}$ holds before i^{th} iteration.

→ Invariance.

Step 1.: Prove $P(1)$

Step 2: Prove $P(i) \rightarrow P(i+1)$ } Induction!

Step 3.: Show correctness at the result.

V. Pseudocodes

- Sorting

```
BubbleSort(A, n) {
   repeat (until no swaps):
      for (int j = 1; j < n-1; j++):
         if A[j] > A[j+1] then
            swap(A, j, j+1)
}
```

```
InsertionSort(A, n):
   for (int i = 1; i < n; i++):
      key = A[i]
      j = i - 1
      while (j >= 0 and A[j] > key) do
         A[j+1] = A[j]
      j--
   A[j+1] = key
```

will be +1 before

```
Selection Sort(A, n):
   for (int j = 0; j < n; j++):
      k = findMin(A, j, n-1)
      swap(A, j, k)
```

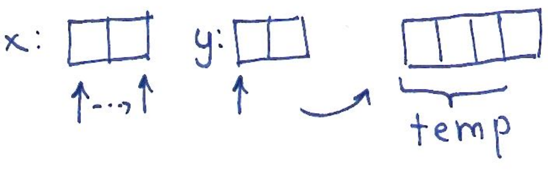
Index of smallest element

```
MergeSort(A, n):
   if (n == 1) return;
   else:
      x = MergeSort(A[1...n/2], n/2);
      y = MergeSort(A[n/2+1...n], n/2)
   return Merge(x, y, n/2)
```

Select smallest element from x or y then increment the pointers (repeat until one array is empty)

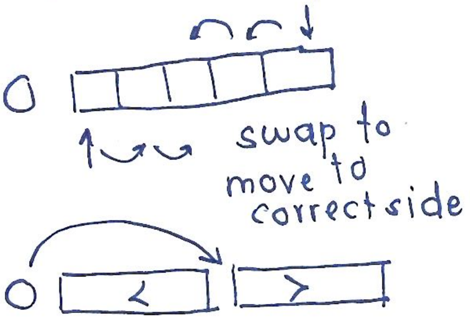
⇓

Repeat the process on array with remaining elements



```
QuickSort(A, n):
    if (n == 1) return
    else:
        p = partition(A, n, pIndex)
        x = QuickSort(A[1...p-1], p-1)
        y = QuickSort(A[p+1...n], n-p)
```

```
partition(A, n, pIndex):
    pivot = A[pIndex]
    swap(A, 1, pIndex)
    low = 2
    high = n+1
    while (low < high) do
        while (A[low] < pivot) and
              (low < high) do low++
        while (A[high] > pivot) and
              (low < high) do high--
        if (low < high) swap(A, low, high)
    swap(A, 1, low-1)
    return low-1
```



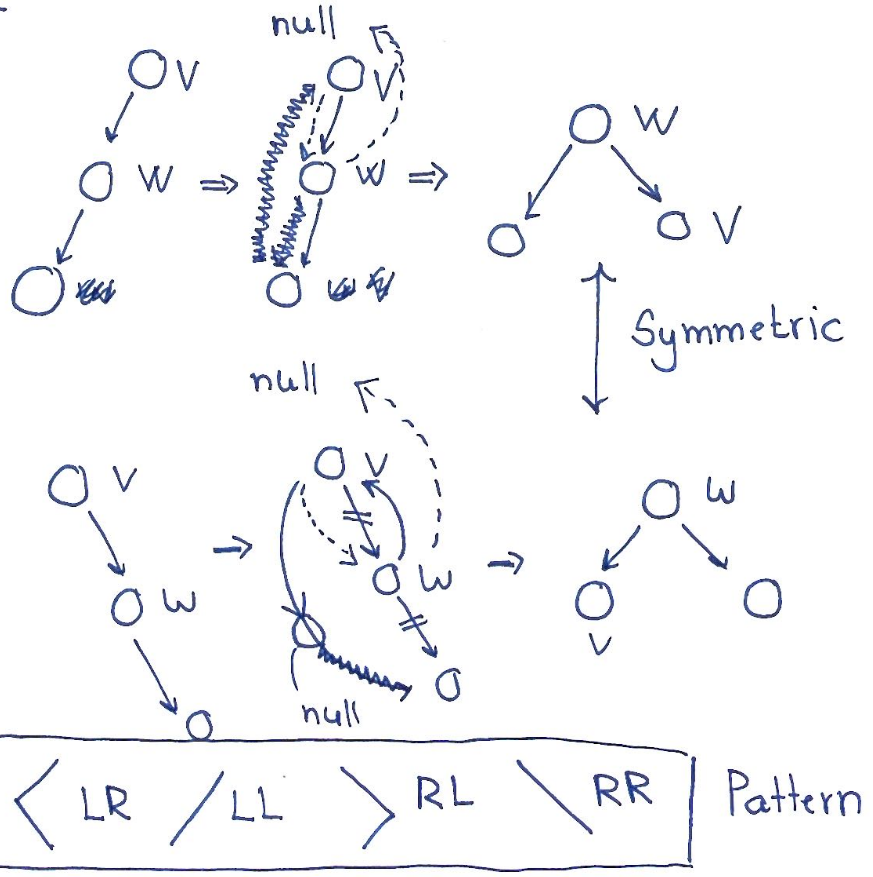
```
QuickSelect(A, n, k):
    if (n==1) return A[1]
    else:
        p = partition(A, n, pIndex);
        if (k == p) return A[p]
        else if (k < p) return QuickSelect(A[1... p-1], k)
        else if (k > p) return QuickSelect(A[~~p+1. p+2... n~~], k-p)
                                             p+1... n
```

randomized

- Rotation in AVL Tree

```
rightRotate(v):
    w = v.left
    w.parent = v.parent
    v.parent = w
    v.left = w.right
    w.right = v
```

```
leftRotate(v):
    w = v.right
    w.parent = v.parent
    v.parent = w
    v.right = w.left
    w.left = v
```



null
v
w
⇒
Symmetric
LR
LL
RL
RR
Pattern