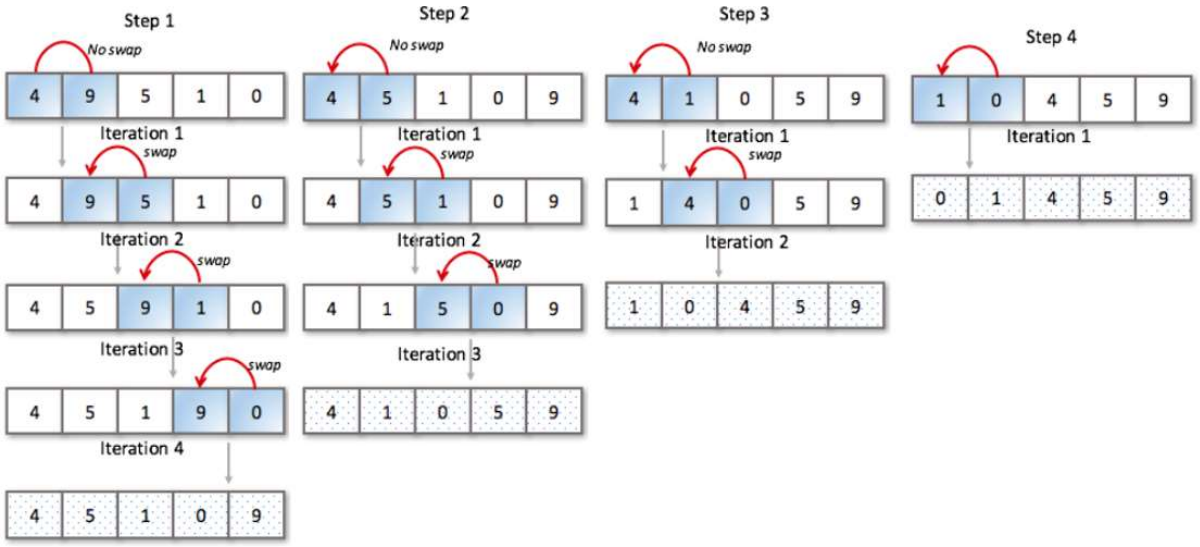
24. 정렬 (Sorting)

1. Bubble Sort

- list 를 처음부터 끝까지 반복 처리하며 인접 item 과 비교하여 위치 교환.
- 매 iteration 마다 가장 큰 숫자가 마지막 element 로 위치
- $O(n^2)$ 의 시간복잡도 (time complexity) 를 가진다.



In [5]:

```
def bubble_sort(arr):
    for i in range(len(arr)):
        for j in range(len(arr)-1-i):    # j 와 j+1 을 비교하므로 -1 을 해 준다.
            if arr[j] > arr[j+1]:        # compare and swap
                arr[j], arr[j+1] = arr[j+1], arr[j]
    return arr
```

In [11]:

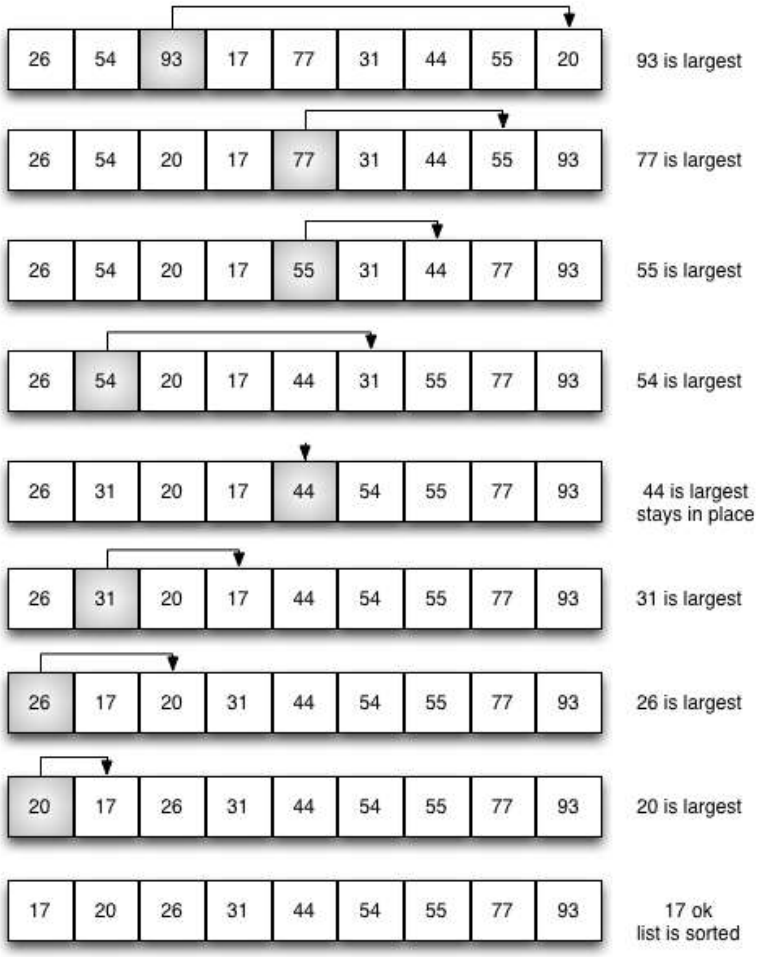
```
import random

%timeit bubble_sort([random.randrange(1, 10000) for _ in range(1000)])
```

86.5 ms ± 3.22 ms per loop (mean ± std. dev. of 7 runs, 10 loops each)

2. Selection Sort (선택정렬)

- bubble sort algorithm 을 개선. 뒤에서부터 index 위치를 거꾸로 내려가며 maxIndex 를 찾아서 한번만 swap 한다.
- $O(n^2)$ 이지만 실질적으로는 bubble sort 보다 빠르다.



In [7]:

```
def selSort(arr):
    for i in range(len(arr)-1, 0, -1):
        maxIndex = 0
        for j in range(i+1):            # i 가 len(arr)-1 부터 시작하므로 +1 을 해주어야 마지막 elem
            if arr[j] > arr[maxIndex]:
                maxIndex = j
        arr[i], arr[maxIndex] = arr[maxIndex], arr[i]
    return arr
```

In [13]:

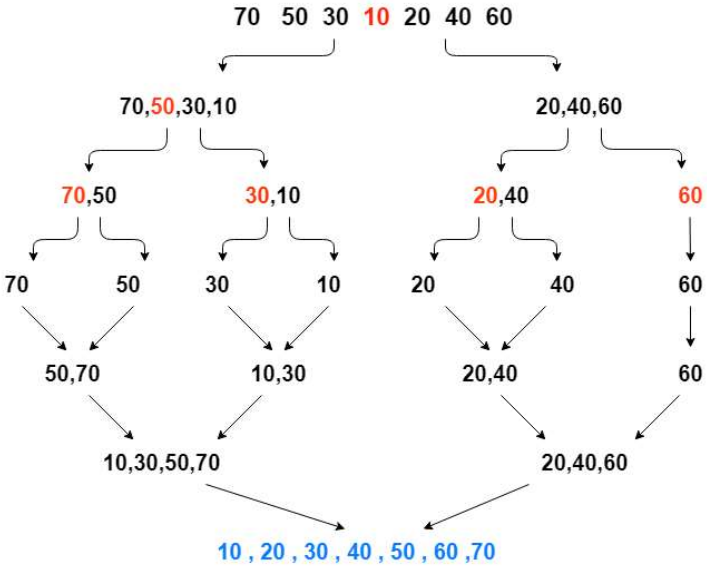
```
%timeit selSort([random.randrange(1, 10000) for _ in range(1000)])
```

33.8 ms ± 2.85 ms per loop (mean ± std. dev. of 7 runs, 10 loops each)

3. MergeSort (병합정렬)

- Divide and Conquer (분할정복) 재귀 알고리즘
- Divide and Conquer 알고리즘의 3 step
  1. 문제를 같은 type 의 문제를 가지는 더 작은 sub-problem 으로 분할한다.
  2. sub-problem 을 재귀적 (recursive) 으로 해결한다 (conquer).
  3. 결과를 recombine (merge) 한다.
- list 의 가운데를 기준으로 좌측 list 와 우측 list 로 분할

- element 가 하나가 될 때까지 list 를 분할 후 좌측과 우측 list element 의 크기를 비교하여 단일 list 로 merge
- 시간복잡도는 O(nLog(n))



In [9]:

```
def mergeSort(arr):
    if len(arr) == 1:       # element 가 하나인 list
        return arr

    a = arr[:int(len(arr)/2)]       # 좌측 절반
    b = arr[int(len(arr)/2):]       # 우측 절반

    a = mergeSort(a)                # c 를 return 하므로 이미 sort 된 list 가 return 됨
    b = mergeSort(b)

    c = []
    i = 0
    j = 0

    while i < len(a) and j < len(b):        # sort 되어 있는 두 개의 list 를 c 로 merge (ex) a - [10,
        if a[i] < b[j]:
            c.append(a[i])
            i += 1
        else:
            c.append(b[j])
            j += 1

    c += a[i:]          # a, b 가 이미 sort 된 상태이므로 단순히 bulk 로 c 에 concatenate
    c += b[j:]
    return c            # sort 된 list return
```

In [14]:

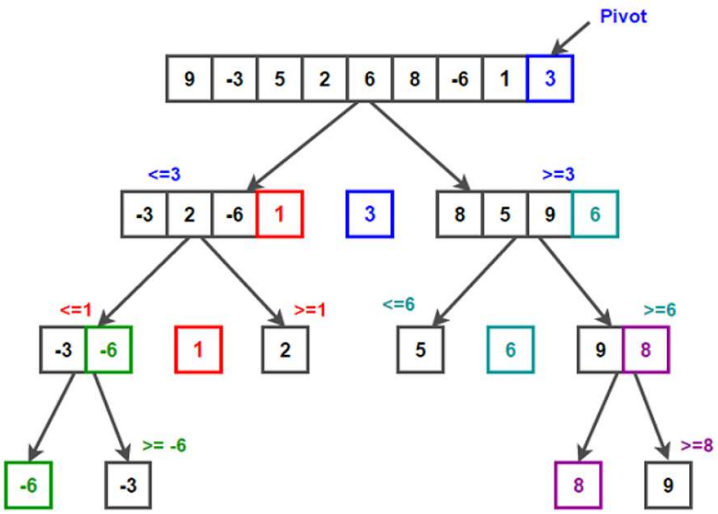
```
%timeit mergeSort([random.randrange(1, 10000) for _ in range(1000)])
```

5.34 ms ± 776 µs per loop (mean ± std. dev. of 7 runs, 100 loops each)

4. Quick Sort (퀵정렬)

- Divide and Conquer (분할정복) 재귀 알고리즘

- list 의 첫번째(혹은 마지막) element 값을 pivit 으로 정하고, 그 보다 작은 sub-list 와 그 보다 큰 sub-list 로 나눈다.
- 각 sub-list 를 남은 원소 갯수가 0 이나 1 이 될 때까지 재귀적으로 나누어 간다.
- 더 이상 나눌 수 없으면 최종 sub-list 를 return 하고 상위 list 에 merge 하여 전체 list 를 연결한다.
- 시간복잡도는 O(nLog(n))
- merge sort 보다 memory 를 적게 사용한다. 단, list 가 반으로 나누어지지 않으면 performance 가 나빠질 수 있다.



In [19]:

```
def qsort(qlist):
    lower = []
    higher = []
    sorted_list = []

    if len(qlist) < 1:
        return          # None returned

    center = qlist[0]                   # 첫번째 element 를 pivot 으로 정함

    for element in qlist[1:]:           # pivot 보다 작으면 lower 에 크면 higher 에 append
        if element <= center:
            lower.append(element)
        else:
            higher.append(element)

    lower = qsort(lower)                # 재귀 호출

    if  lower != None:                  # 이미 sort 된 lower 부분이 return 되므로 단순히 concatenate
        sorted_list += lower

    sorted_list.append(center)          # pivot (중간) element 를 append

    higher = qsort(higher)              # 이미 sort 된 higher 부분이 return 되므로 단순히 concatenat
    if  higher  != None:
        sorted_list += higher

    return sorted_list                  # concatenate 된 list 반환
```

In [20]:

```
%timeit qsort([random.randrange(1, 10000) for _ in range(1000)])
```

3.45 ms ± 479 µs per loop (mean ± std. dev. of 7 runs, 100 loops each)

연습문제

- 정렬할 list 의 마지막 element 를 pivot 으로 정하는 quick sort 함수 작성

In [17]:

```
def qsort(qlist):

    #CODE HERE

    return sorted_list                    # concatenate 된 list 반환
```

Python 내장 sort 함수

- list.sort()
- sorted()

list 정렬

In [2]:

```
a = [1, 10, 5, 7, 6]
a.sort()

print(a)
```

[1, 5, 6, 7, 10]

In [3]:

```
sorted([5, 2, 3, 1, 4])
```

Out[3]:

[1, 2, 3, 4, 5]

In [4]:

```
sorted("This is a test string from Andrew".split(), key=str.lower)
```

Out[4]:

['a', 'Andrew', 'from', 'is', 'string', 'test', 'This']

In [6]:

```
student_tuples = [
    ('john', 'A', 15),
    ('jane', 'B', 12),
    ('dave', 'B', 10),
]

sorted(student_tuples, key=lambda student: student[2])   # sort by age
```

Out[6]:

```
[('dave', 'B', 10), ('jane', 'B', 12), ('john', 'A', 15)]
```

In [7]:

```
sorted(student_tuples, key=lambda student: student[2], reverse=True)
```

Out[7]:

```
[('john', 'A', 15), ('jane', 'B', 12), ('dave', 'B', 10)]
```

dictionary 정렬

In [19]:

```
dict = {'A' :100,'D' :95,'C' :80,'B' :70}

sorted(dict.items(), key=lambda kv: kv[0], reverse=True)
```

Out[19]:

```
[('D', 95), ('C', 80), ('B', 70), ('A', 100)]
```

In [21]:

```
sorted(dict.items(), key=lambda kv: kv[1])
```

Out[21]:

```
[('B', 70), ('C', 80), ('D', 95), ('A', 100)]
```

연습문제

학생 list 를 우수한 성적순으로 정렬

In [14]:

```
students = [
        ('홍길동', 3.9, 2016303),
        ('김철수', 3.0, 2016302),
        ('최자영', 4.3, 2016301),
]
```

In [15]:

```
# CODE HERE
```

Out[15]:

[('최자영', 4.3, 2016301), ('홍길동', 3.9, 2016303), ('김철수', 3.0, 2016302)]

dictionary 를 value 순으로 정렬

In [18]:

```
dict = {'A' :5,'D' :7,'C' :3,'B' :2}

# CODE HERE
```

Out[18]:

[('B', 2), ('C', 3), ('A', 5), ('D', 7)]