MYONGJI UNIVERSITY

Sort

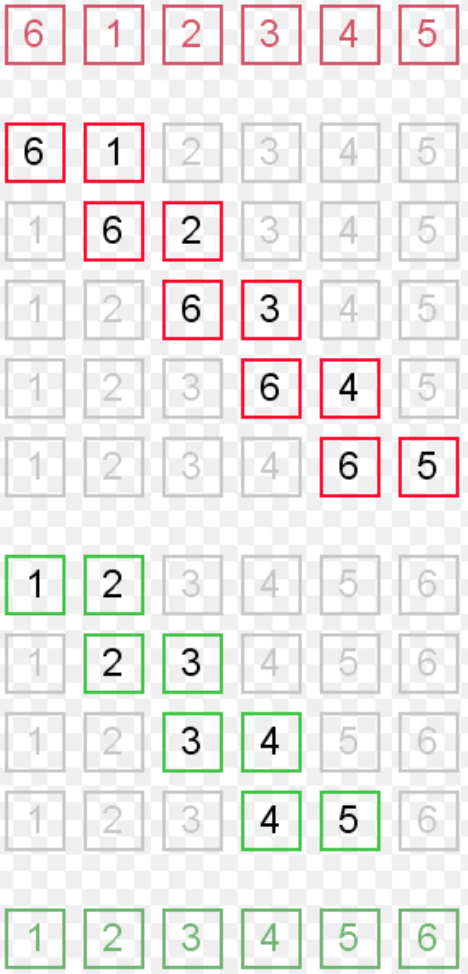
Dr. Seung Chul Han
Dept. Computer Engineering
Myongji University

Sort

- 정렬
- 주어진 집합의 원소들을 순서대로 나열
- 컴텨에서 가장 많이 사용
- Heap, selection, bubble, tree, insertion, merge, shell, radix, quick etc.

Bubble Sort

- 이웃한 2원소를 정렬

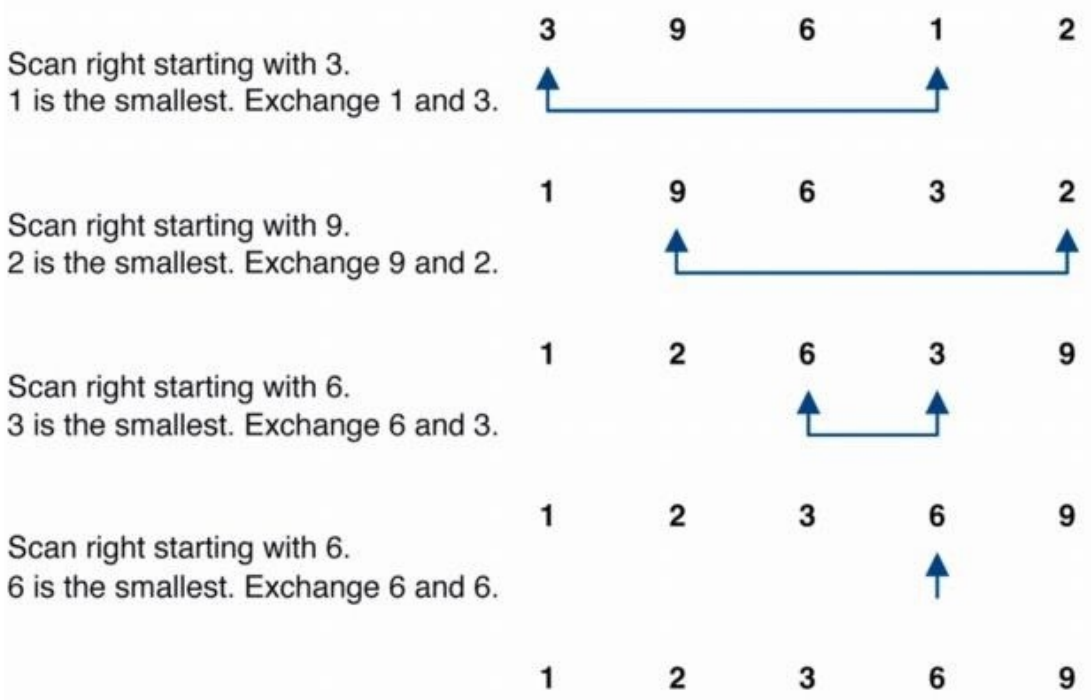



```
void bubblemax(int x[], int eff_size)
{    int k;

     for (k = 0; k < eff_size - 1; k++)
          if (x[k] > x[k + 1])
               swaparay(x, k, k + 1);
}

void swaparay(int x[], int i, int j)
{    int temp;

     temp = x[i];
     x[i] = x[j];
     x[j] = temp;
}
```

최악의 경우 n^2회 swap: $O(n^2)$

Selection Sort

- *k*번째h smallest 찾기



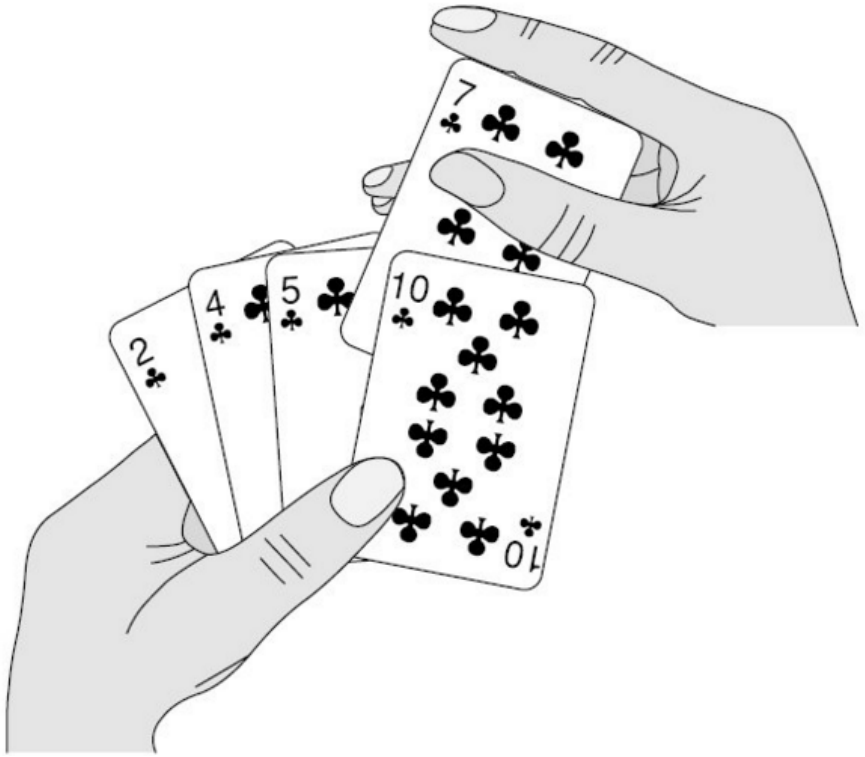

```
void SelectionSort(int A[], int n)
 {
    int i, j, small, temp;
(1) for (i = 0; i < n-1; i++) {
(2)   small = i;
(3)   for (j = i+1; j < n; j++)
(4)     if (A[j] < A[small])
(5)     small = j;
(6) temp = A[small];
(7) A[small] = A[i];
(8) A[i] = temp;
     }
    }
```

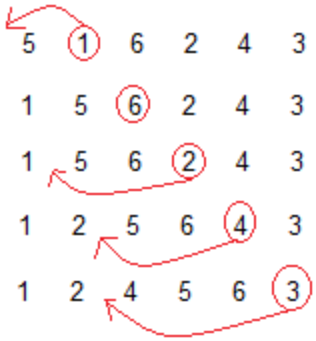
최악의 경우 n^2회 swap: $O(n^2)$

Insertion Sort

- 한 원소를 최대한 왼쪽으로

insertion





(Always we start with the second element as key.)

As we can see here, in insertion sort, we pick up a key, and compares it with elemnts ahead of it, and puts the key in the right place

5 has nothing before it.

1 is compared to 5 and is inserted before 5.

6 is greater than 5 and 1.

2 is smaller than 6 and 5, but greater than 1, so its is inserted after 1.

And this goes on...

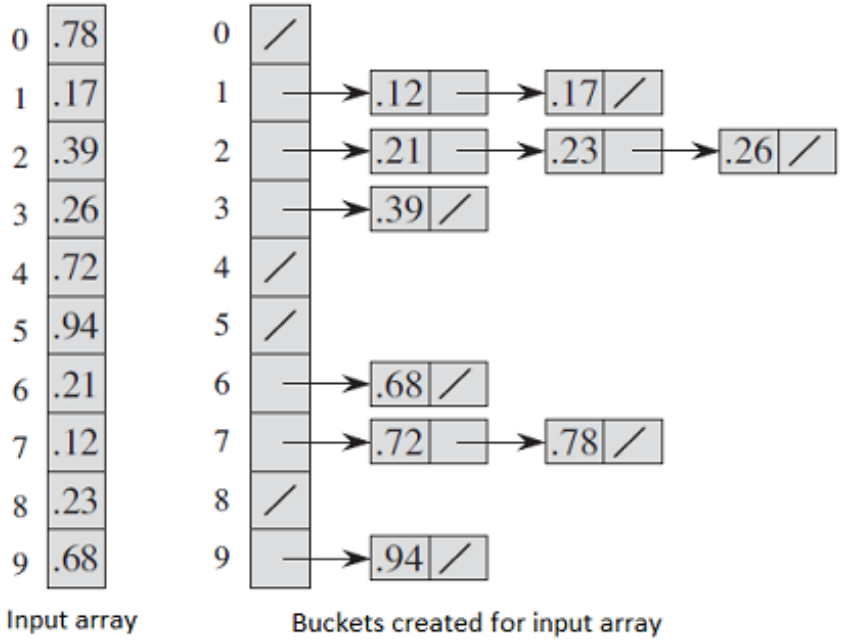
```
InsertionSort(A, n) {
  for i = 2 to n {
      key = A[i]
      j = i - 1;
      while (j > 0) and (A[j] > key) {
            A[j+1] = A[j]
            j = j - 1
      }
      A[j+1] = key
  }
}
```

최악의 경우 n^2회 swap: $O(n^2)$

Bucket Sort

- 데이터가 uniform distribute일때 유용

```
bucketSort(arr[], n)
1) n 개의 빈 bucket(list)을 생성한다.
2) arr[]의 모든 원소들을 n으로 나누어 bucket list에 골고루 들어가게 한다. 같은bucket안의 들어가는 원소들은 다시 list로 보관한다.
3) 각 bucket안의 list에 있는 원소들을 sorting한다.
4) bucket간에는 정렬이 되어 있는 상태이므로 이들을 순차적으로 활용하면 전체가 정렬된다.
```



Input array

Buckets created for input array

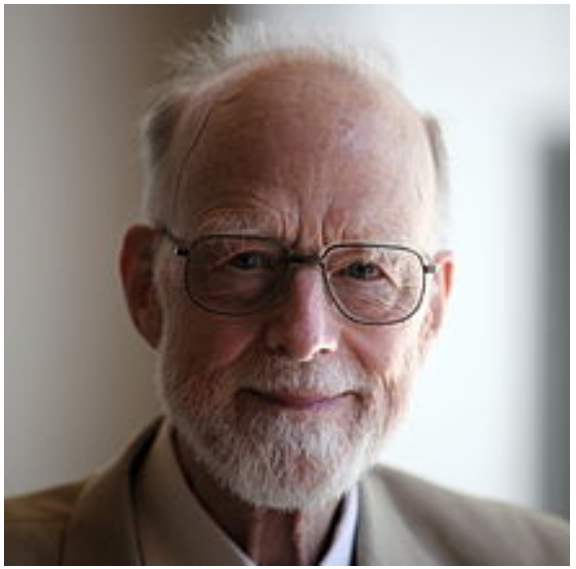
```cpp
#include <stdio.h>
#include <algorithm>
#include <vector>
using namespace std;

// Function to sort arr[] of size n using bucket sort
void bucketSort(float arr[], int n)
{
    // 1) Create n empty buckets
    vector<float> b[n];

    // 2) Put array elements in different buckets
    for (int i = 0; i < n; i++)
    {
        int bi = n*arr[i]; // Index in bucket
        b[bi].push_back(arr[i]);
    }

    // 3) Sort individual buckets
    for (int i = 0; i < n; i++)
        sort(b[i].begin(), b[i].end());

    // 4) Concatenate all buckets into arr[]
    int index = 0;
    for (int i = 0; i < n; i++)
        for (int j = 0; j < b[i].size(); j++)
            arr[index++] = b[i][j];
}

/* Driver program to test above funtion */
int main()
{
    float arr[] = { 0.897, 0.565, 0.656, 0.1234, 0.665, 0.3434 };
    int n = sizeof(arr) / sizeof(arr[0]);
    bucketSort(arr, n);

    printf("Sorted array is \n");
    for (int i = 0; i < n; i++)
        printf("%f ", arr[i]);

    return 0;
}
```

Practice

넥센 대 LG 상대 전적													
넥센							LG						
	점	타율	방어	선발	승패투수	세이브		점	타율	방어	선발	승패투수	세이브
원	0	0.172	12.38	신재영	신재영(패)		홈	11	0.400	0.00	류제국	류제국(승)	
원	2	0.235	5.63	박주현	오주원(패)		홈	5	0.286	2.00	봉중근	김지용(승)	임정우
홈	2	0.212	3.00	신재영	신재영(패)		원	3	0.265	2.00	류제국	류제국(승)	임정우
홈	4	0.314	9.00	최원태	최원태(패)		원	9	0.378	4.00	허프	허프(승)	
홈	7	0.313	4.00	신재영	이보근(승)	김세현	원	4	0.257	7.88	허프	진해수(패)	
홈	7	0.281	3.00	박주현	김상수(승)		원	3	0.286	6.75	우규민	이동현(패)	
홈	6	0.278	12.00	피어밴	이보근(패)		원	12	0.357	6.00	소사	유원상(승)	
원	1	0.200	2.25	맥그레	맥그레(패)		홈	2	0.207	1.00	류제국	류제국(승)	신승현
원	8	0.342	5.40	최원태	김세현(승)	오재영	홈	6	0.300	5.40	소사	임정우(패)	
원	7	0.250	10.13	금민철	이보근(패)		홈	9	0.400	7.00	장진용	이동현(승)	임정우
원	4	0.286	4.50	신재영	김택형(패)		홈	5	0.241	4.00	소사	소사(승)	임정우
원	7	0.351	2.00	양훈	하영민(승)	김세현	홈	5	0.333	6.00	우규민	우규민(패)	
원	3	0.143	4.50	박주현	이보근(패)		홈	4	0.310	3.00	코프랜	이승현(승)	임정우
홈	3	0.306	5.00	피어밴	김상수(패)		원	5	0.294	3.00	소사	이동현(승)	임정우
홈	14	0.450	1.00	신재영	신재영(승)		원	2	0.222	15.75	류제국	류제국(패)	
홈	10	0.400	2.00	박주현	박주현(승)		원	2	0.161	10.13	코프랜	코프랜(패)	

Quick Sort

- Sir Tony Hoare, 1959
- 가장 많이 사용
- 최악은 $O(n^2)$이지만, 평균 $O(nlogn)$

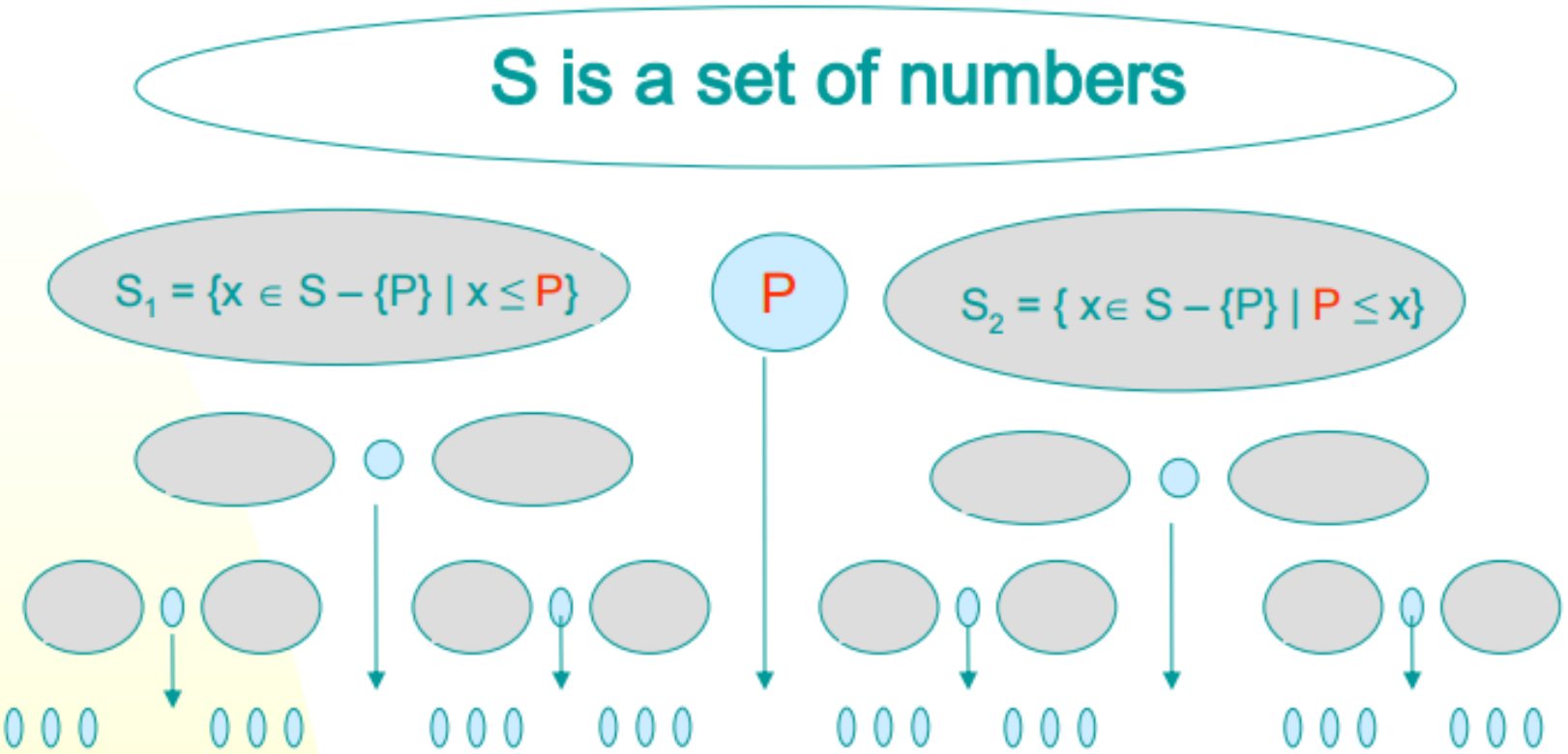


Basic Ideas

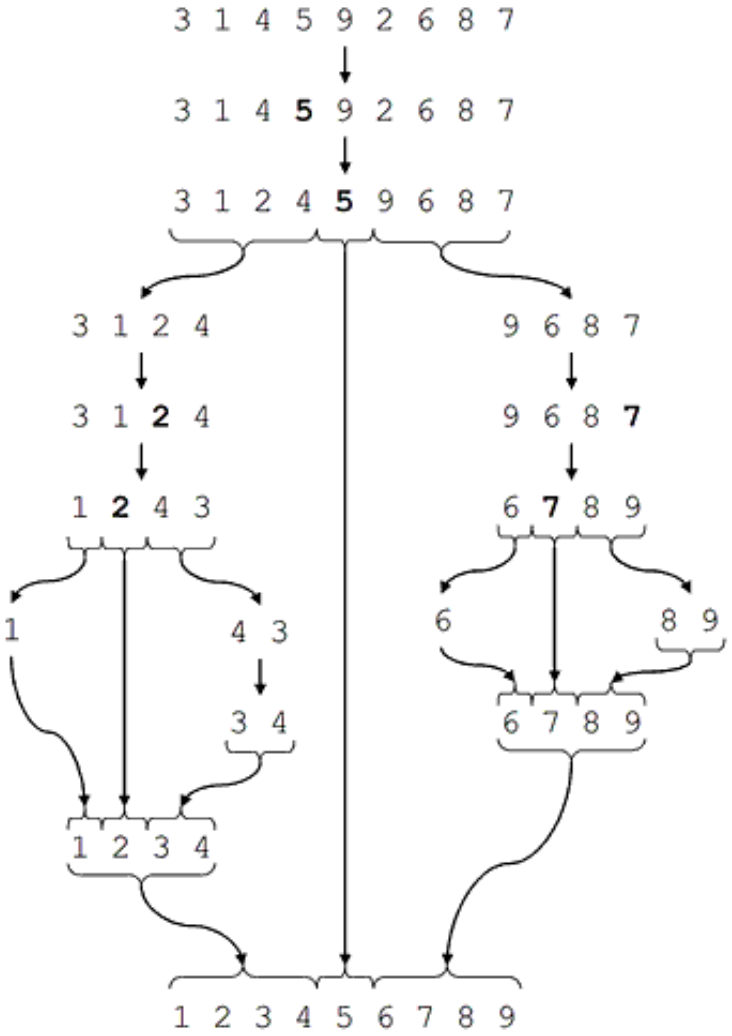
(Another divide-and-conquer algorithm)

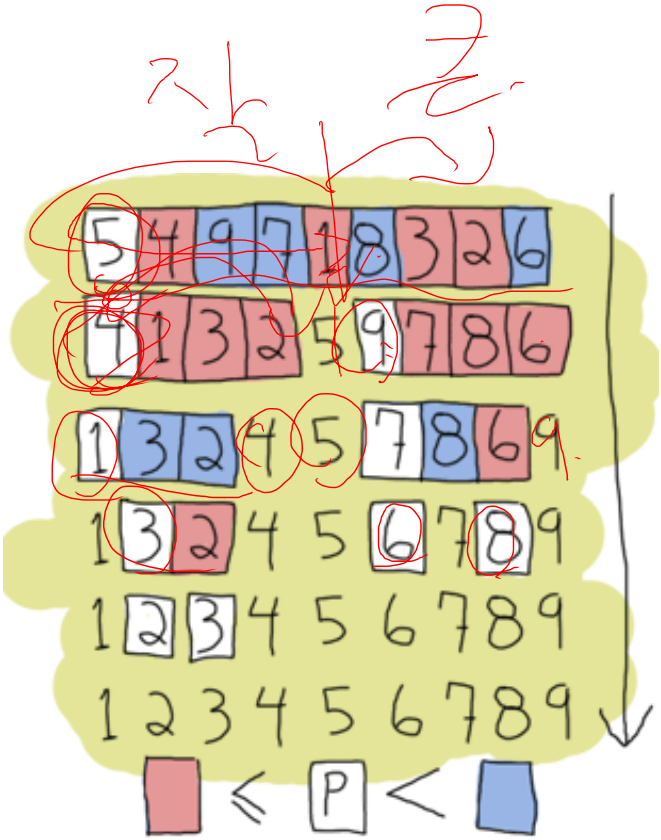
- Pick an element, say ***P*** (the pivot)
- Re-arrange the elements into 3 sub-blocks,
  1. those less than or equal to (≤) ***P*** (the **left-**block S_1)
  2. ***P*** (the only element in the **middle**-block)
  3. those greater than or equal to (≥) ***P*** (the **right-**block S_2)
- Repeat the process **recursively** for the **left-** and **right-** sub-blocks. Return {quicksort(S_1), ***P***, quicksort(S_2)}. (That is the results of quicksort(S_1), followed by ***P***, followed by the results of quicksort(S_2))

Basic Ideas



Example


3 1 4 5 9 2 6 8 7
3 1 4 5 9 2 6 8 7
3 1 2 4 5 9 6 8 7
3 1 2 4
9 6 8 7
3 1 2 4
9 6 8 7
1 2 4 3
6 7 8 9
1
4 3
6
8 9
3 4
6 7 8 9
1 2 3 4
1 2 3 4 5 6 7 8 9


P

Algorithm

```
void quickSort( int[], int, int);
int partition( int[], int, int);


void main()
{
        int a[] = { 7, 12, 1, -2, 0, 15, 4, 11, 9};

        int i;
        printf("\n\nUnsorted array is:  ");
        for(i = 0; i < 9; ++i)
                printf(" %d ", a[i]);

        quickSort( a, 0, 8);

        printf("\n\nSorted array is:  ");
        for(i = 0; i < 9; ++i)
                printf(" %d ", a[i]);

}
```

```
void quickSort( int a[], int l, int r)
{
   int j;

   if( l < r )
   {
        // divide and conquer
        j = partition( a, l, r);
       quickSort( a, l, j-1);
       quickSort( a, j+1, r);
   }

}



int partition( int a[], int l, int r) {
   int pivot, i, j, t;
   pivot = a[l];
   i = l; j = r+1;

   while( 1)
   {
        do ++i; while( a[i] <= pivot && i <= r );
        do --j; while( a[j] > pivot );
        if( i >= j ) break;
        t = a[i]; a[i] = a[j]; a[j] = t;
   }
   t = a[l]; a[l] = a[j]; a[j] = t;
   return j;
}
```

Pivot을 잘 못고르면
최악의 경우 n^2회 swap: $O(n^2)$

Quiz. cs.nyu.edu/web/Academic/Graduate/exams

Given the array $A[1,\ldots,8] =< 6,10,13,5,8,3,2,11 >$, the pivot element $A[1] = 6$, and the Partition pseudo-code

```
PARTITION(A, p, q):
    1.  let x ← A[p]    (pivot)
    2.  let i ← p
    3.  for j ← p + 1 to q
    4.    do if A[j] ≤ x,
    5.         then i ← i + 1
    6.              exchange A[i] ↔ A[j]
    7.              print A
    8.  exchange A[i] ↔ A[j]
    9.  print A
    10. return i
```

Show the arrays that are printed eachtime lines 7 and 9 are executed (for a total of 4 arrays).

1. $A =< 6,5,10,13,8,3,2,11 >$
2. $A =< 6,5,3,10,13,8,2,11 >$
3. $A =< 6,5,3,2,10,13,8,11 >$
4. $A =< 5,3,2,6,10,13,8,11 >$

Quiz.
http://www.cs.cornell.edu/courses/cs2110/2016sp/

1.(20 points) Sort Algorithms

(a) *(6 points)* There are three fundamental steps to the Quicksort algorithm. What are they?

1. Choose a pivot
2. Separate the elements into two sets: Greater and less than the pivot
3. Recursively Quicksort each of these sets

(b) *(5 points)* Quicksort has an average case runtime of $O(n \log n)$, but a worst case complexity of $O(n^2)$. Is this an issue in practice? Explain.

Not really. We want the pivot value to split the input vector into two equal size halves. The usual pivot is to select the value in the middle of the vector: if the input was sorted, this will split the vector; if not, it should be a pretty random value. Performance can still be poor for a hand-crafted "evil vector" but the odds of hitting such a vector of inputs "in the wild" should be pretty low.

Quiz. www.cs.colorado.edu/~main/questions/

8. Here is an array of ten integers:

5 3 8 9 1 7 0 2 6 4

Suppose we partition this array using quicksort's partition function and using 5 for the pivot. Draw the resulting array after the partition finishes.

Quicksort가 worst case가 되는 경우는?

HOMEWORK

- Bubble, Selection, Insertion, Quick Sorting 방법들을 다음의 데이터에 따라 수행시간비교 그래프 그릴것.
- int a[10], int a[100], int a[1000], int a[10000], int a[100000] 그 이상은 자유
- Internet open source 사용가능하지만, 최대한 직접 coding

- * a[ ]의 데이터는 랜덤으로 생성

Find *k*th smallest number

- Unsorted set of integer, find *k*th smallest number
- 무식한 Algorithm

- Sort하고, k번째 select

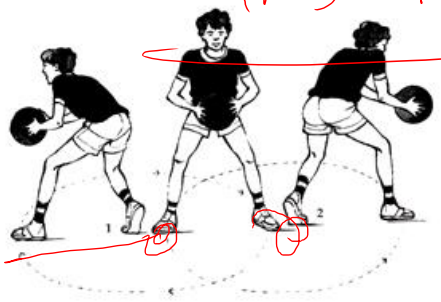
How about this?

Algorithm

1. Select a pivot and put that in its correct position
2. Now check if the pivot is at K-th position, If yes then return Pivot position
3. If pivot position is less than K, then desired element is right sub-array , repeat step 1 and 2 on right sub-array
4. If pivot position is greater than pivot position, then desired element is in left sub-array; repeat step 1 and 2 on left sub array.

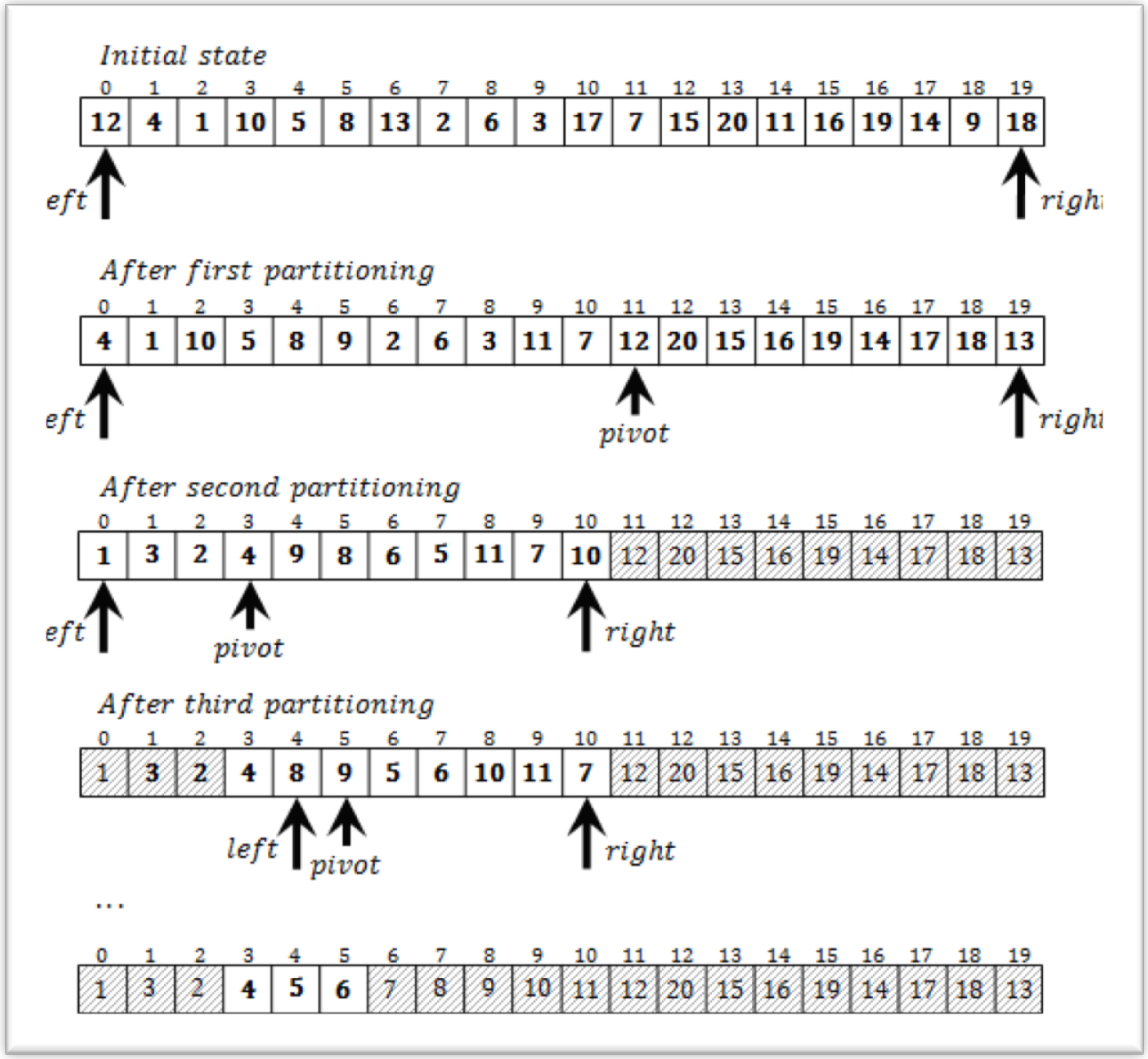
Bubub
Bubble
selection
quck
insertion

Pivot: 회전축



Unsorted case: kth smallest 찾기

- 정렬되지 않은 n개의 정수가 있을때, k번째 작은 수 찾기



- Worst : O(n)
- Average : O(logn)