프로그래밍언어

6. 배열 기반의 기본 알고리즘 (탐색, 정렬), 성능 측정

교수 김 영 탁

영남대학교 정보통신공학과

(Tel : +82-53-810-2497; Fax : +82-53-810-4742

http://antl.yu.ac.kr/; E-mail : ytkim@yu.ac.kr)

Outline

- 배열 기반 기본 알고리즘 – 탐색, 정렬
- 순차탐색 (Sequential Search)
- 이진탐색 (Binary Search)
- 선택정렬 (Selection Sorting)
- 퀵 정렬 (Quick Sorting)
- 병합정렬 (Merge Sorting)

- 성능측정 및 분석
  - Windows 운영체제의 Query Performance Counter 기능을 사용한 milli-second/micro-second 단위 실행시간 정밀측정

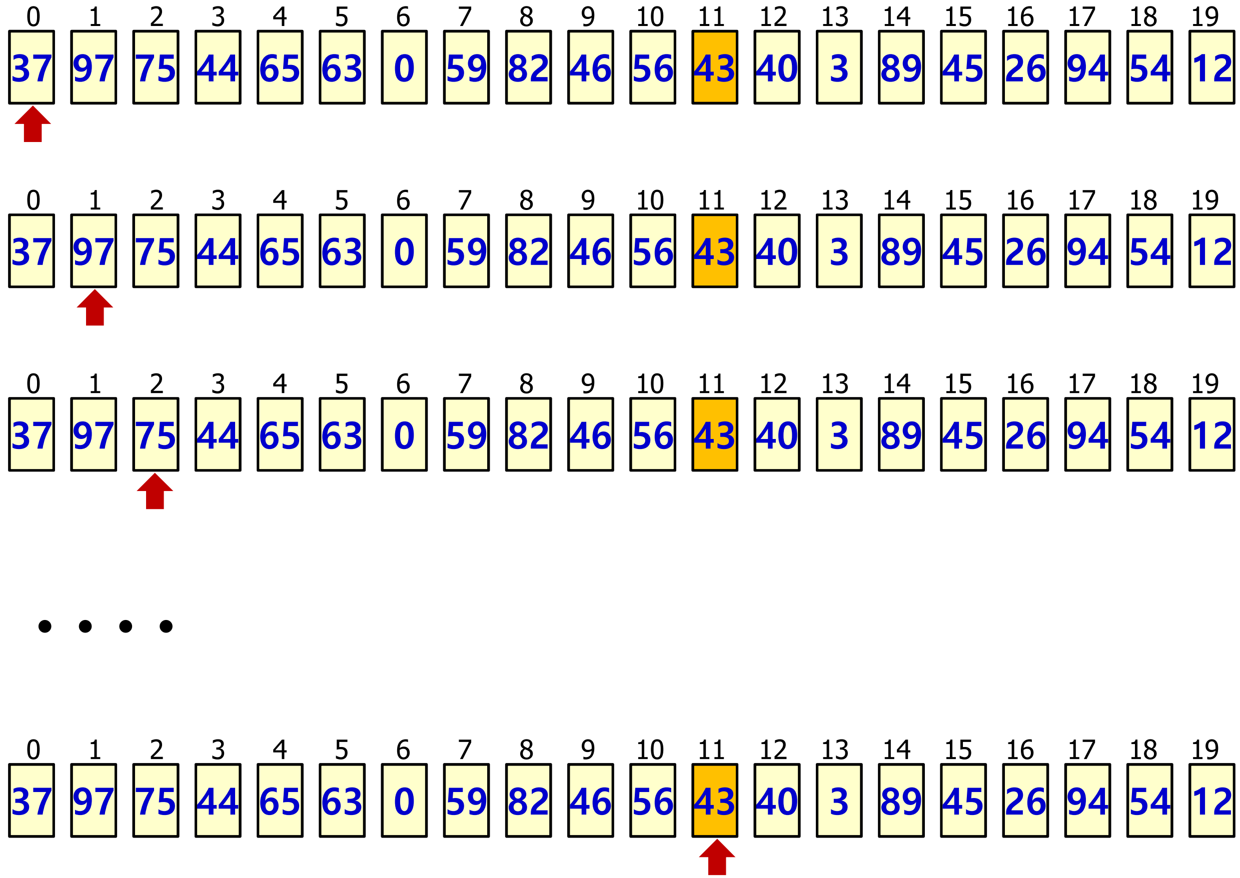
순차 탐색 (Sequential Searching)

순차탐색 (Sequential Search)

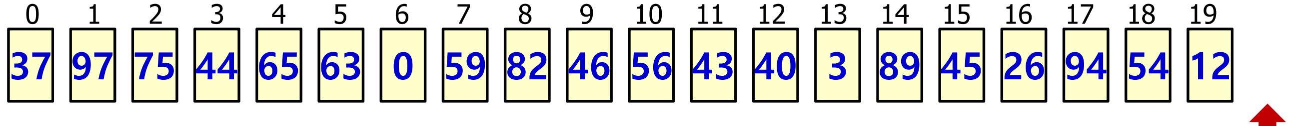
◆ **순차 탐색은 배열의 원소를 순서대로 하나씩 꺼내서 탐색키와 비교하여 원하는 값을 찾아가는 방법**

- 배열에 포함된 원소가 정렬되어 있지 않을 수 있음
- 가장 빠른 탐색 결과: 맨 처음 원소가 찾고자 하는 원소일 때
- 가장 늦은 탐색 결과: 맨 뒤 원소가 찾고자 하는 원소 일 때
- 만약 찾고자 하는 원소가 배열에 포함되어 있지 않을 때 : -1 반환
- N개의 원소가 포함된 배열에서의 탐색에 걸리는 시간의 평균: N/2

순차탐색



탐색키워드가 존재할 때 (해당 **position** 반환)



탐색키워드 (예: **25**)가 존재하지 않을 때 (**-1** 반환)

BigArray.h - 헤더파일

```
/* BigArray.h*/

#ifndef BIG_ARRAY_H
#define BIG_ARRAY_H

#include <stdio.h>

void printArray(int *array, int size, int line_size);
void fprintArray(FILE *fout, int *array, int size, int line_size);
void printBigArraySample(int *array, int size, int items_per_line, int num_sample_lines);
void fprintBigArraySample(FILE *fout, int *array, int size,
        int items_per_line, int num_sample_lines);
void genBigRandArray(int *array, int size, int base);
void suffleArray(int *array, int size);
int sequentialSearch(int *array, int size, int key_to_search);
int binarySearch(int *array, int size, int key);
void selectionSort(int *array, int size);
void quickSort(int *array, int size);
void getArrayStatistics(int *array, int size);
void fgetArrayStatistics(FILE *fout, int *array, int size);
#endif
```

sequentialSearch()

```
/* BigArray.cpp */

int sequentialSearch(int *array, int size, int key_to_search)
{
    for (int pos = 0; pos < size; pos++)
    {
        if (array[pos] == key_to_search)
        {
            return pos;
        }
    }
    return -1;
}
```

main()

```
/* main() for Algorithms_on_Arrays (1)  */
#include <stdio.h>
#include <conio.h>
#include <stdlib.h>
#include <time.h>
#include <Windows.h>
#include "BigArray.h"
#include "String_Algorithms.h"

#define ESC 0x1B
#define NUM_DATA 20
#define LINE_SIZE 20

// prototypes of functions used in main()
void testSequentialSearch(FILE *fout);
void testBinarySearch(FILE *fout);
void testSelectionSort(FILE *fout);
void testQuickSort(FILE *fout);
void testSelectionSort_Words(FILE *fout);
void testQuickSort_Words(FILE *fout);
void PM_QuickSort_IntArray(FILE *fout);
void PM_SelectionSort_IntArray(FILE *fout);
```

```
/* main() for Algorithms_on_Arrays (2)  */
int main()
{
	FILE *fout;
	char menu;

	fout = fopen("output.txt",  "w");
	if (fout == NULL)
	{
		printf("Error in creation of array_output.txt !!\n");
		return -1;
	}

	while (1)
	{
		printf("\nTest Array Algorithms :\n");
		printf("   1: Test Sequential Search\n");
		printf("   2: Test Binary Search\n");
		printf("   3: Test Selection Sort\n");
		printf("   4: Test Quick Sort\n");
		printf("   5: Test Selection Sort for Words\n");
		printf("   6: Test Quick Sort for Words\n");
		printf("   7: Performance Measurements of Quick Sort for Integer Array\n");
		printf("   8: Performance Measurements of Selection Sort for Integer Array\n");
		printf("   Esc: terminate\n");
		printf("Input menu : ");
		menu = getchar();
		printf("\n");
		if (menu == ESC)
			break;
```

```
/* main() for Algorithms_on_Arrays (3)  */

            switch (menu)
            {
            case '1':
                  testSequentialSearch(fout);
                  break;
            case '2':
                  testBinarySearch(fout);
                  break;
            case '3':
                  testSelectionSort(fout);
                  break;
            case '4':
                  testQuickSort(fout);
                  break;
            case '5':
                  testSelectionSort_Words(fout);
                  break;
            case '6':
                  testQuickSort_Words(fout);
                  break;
            case '7':
                  PM_QuickSort_IntArray(fout);
                  break;
            case '8':
                  PM_SelectionSort_IntArray(fout);
                  break;
            default:
                  break;
            }
      }
      fclose(fout);
      return 0;
}
```

```
void testSequentialSearch(FILE *fout)
{
	int data_array[NUM_DATA] =
	    { 37, 97, 75, 44, 65, 63, 0, 59, 82, 46, 56, 43, 40, 3, 89, 45, 26, 94, 54, 12 };
	int key_to_search;
	int pos;

	printf("Integer array to be searched:\n");
	printArray(data_array, NUM_DATA, LINE_SIZE);
	while (1)
	{
		printf("Input integer to be searched (-1 to quit) : ");
		scanf("%d", &key_to_search);
		if (key_to_search == -1)
			break;
		else
		{
			printf("Sequential searching key (%d) from array of %d data ... \n",
			    key_to_search, NUM_DATA);
			pos = sequentialSearch(data_array, NUM_DATA, key_to_search);
			if (pos == -1)
			{
				printf("The key (%2d) was not found from the array\n", key_to_search);
			}
			else
			{
				printf("The key (%2d) was found at position (%2d)\n", key_to_search, pos);
			}
		}
	}
}
```

SequentialSearch() 실행결과

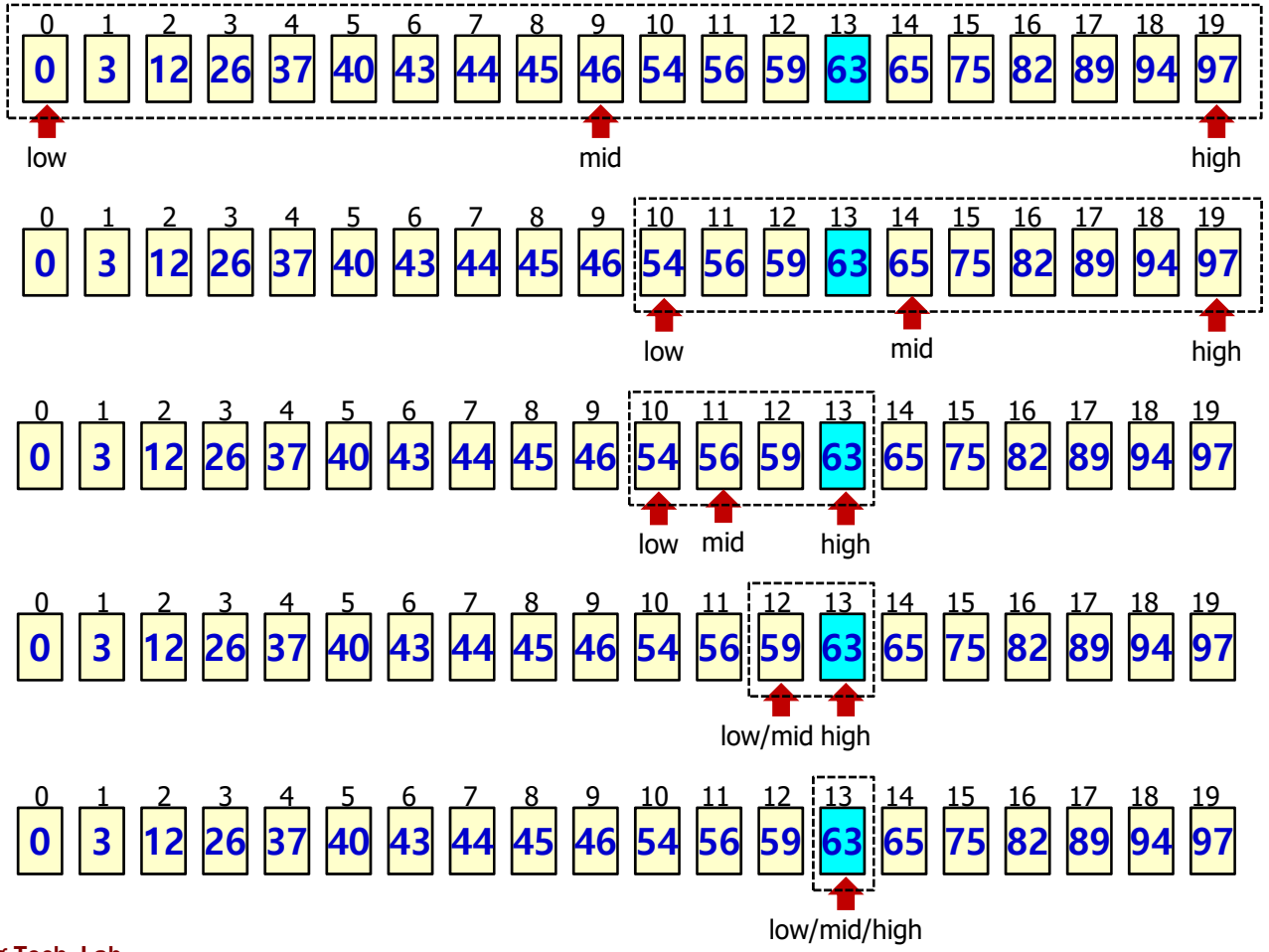
```
Test Array Algorithms :
   1: Test Sequential Search
   2: Test Binary Search
   3: Test Selection Sort
   4: Test Quick Sort
   5: Test Selection Sort for Words
   6: Test Quick Sort for Words
   7: Performance Measurements of Quick Sort for Integer Array
   8: Performance Measurements of Selection Sort for Integer Array
   Esc: terminate
Input menu : 1

Integer array to be searched:
   37   97   75   44   65   63    0   59   82   46   56   43   40    3   89   45   26   94   54   12
Input integer to be searched (-1 to quit) : 12
Sequential searching key (12) from array of 20 data ...
The key (12) was found at position (19)
Input integer to be searched (-1 to quit) : 0
Sequential searching key (0) from array of 20 data ...
The key ( 0) was found at position ( 6)
Input integer to be searched (-1 to quit) : 37
Sequential searching key (37) from array of 20 data ...
The key (37) was found at position ( 0)
Input integer to be searched (-1 to quit) :
```

이진 탐색 (Binary Searching)

이진 탐색 (binary search)

◆ 이진 탐색(binary search)

- 사전에 정렬되어 있는 배열의 중앙에 위치한 원소와 비교 되풀이



binarySearch()

```
int binarySearch(int *array, int size, int key_to_search)
{
	int low, high, mid;
	int loop = 0;

	if (key_to_search > array[size - 1])
	{
		printf("Binary_Search :: given key (%d) is beyond the maximum value of the
		    array (%d)\n", key_to_search, array[size - 1]);
		return -1;
	}
	low = 0;
	high = size - 1;
	loop++;
	while (low <= high)
	{
		printf("%2d-th loop: Search range: [%2d ~ %2d]\n", loop, low, high);
		mid = (low + high) / 2;
		if (key_to_search == array[mid])
			return mid;
		else if (key_to_search < array[mid])
			high = mid - 1;
		else
			low = mid + 1;
		loop++;
	}
	return -1;
}
```

main()

```
/* main.cpp */
void testBinarySearch(FILE *fout)
{
	int data_array[NUM_DATA] =
	   { 0, 3, 12, 26, 37, 40, 43, 44, 45, 46, 54, 56, 59, 63, 66, 75, 82, 89, 94, 97 };

	int key_to_search;
	int pos;

	printf("Integer array to be searched:\n");
	printArray(data_array, NUM_DATA, LINE_SIZE);
	while (1)
	{
		printf("Input integer to be searched (-1 to quit) : ");
		scanf("%d", &key_to_search);
		if (key_to_search == -1)
			break;
		else
		{
			printf("Binary searching key (%d) from array of %d data ... \n",
			   key_to_search, NUM_DATA);
			pos = binarySearch(data_array, NUM_DATA, key_to_search);
			if (pos == -1)
			{
				printf("The key (%2d) was not found from the array\n", key_to_search);
			}
			else
			{
				printf("The key (%2d) was found at position (%2d)\n", key_to_search, pos);
			}
		}
	}
}
```

binary_search() 실행 결과

```
Test Array Algorithms :
   1: Test Sequential Search
   2: Test Binary Search
   3: Test Selection Sort
   4: Test Quick Sort
   5: Test Selection Sort for Words
   6: Test Quick Sort for Words
   7: Performance Measurements of Quick Sort for Integer Array
   8: Performance Measurements of Selection Sort for Integer Array
   Esc: terminate
Input menu : 2

Integer array to be searched:
    0    3   12   26   37   40   43   44   45   46   54   56   59   63   66   75   82   89   94   97
Input integer to be searched (-1 to quit) : 97
Binary searching key (97) from array of 20 data ...
 1-th loop: Search range: [ 0 19]
 2-th loop: Search range: [10 19]
 3-th loop: Search range: [15 19]
 4-th loop: Search range: [18 19]
 5-th loop: Search range: [19 19]
The key (97) was found at position (19)
Input integer to be searched (-1 to quit) : 46
Binary searching key (46) from array of 20 data ...
 1-th loop: Search range: [ 0 19]
The key (46) was found at position ( 9)
Input integer to be searched (-1 to quit) : 0
Binary searching key (0) from array of 20 data ...
 1-th loop: Search range: [ 0 19]
 2-th loop: Search range: [ 0  8]
 3-th loop: Search range: [ 0  3]
 4-th loop: Search range: [ 0  0]
The key ( 0) was found at position ( 0)
Input integer to be searched (-1 to quit) : -1
```

순차 탐색과 이진 탐색의 비교

◆ **N개의 원소를 가진 배열에서 지정된 값을 탐색하기 위한 비교 (comparison) 횟수**

- **순차 탐색**
  - 주어진 배열이 **정렬되어 있지 않아도 사용 가능**
  - 최소: 1 (첫 번째 원소가 탐색 대상일 경우)
  - 최대: N (마지막 원소가 탐색 대상일 경우)
  - 평균: N/2

- **이진 탐색**
  - 주어진 배열이 반드시 **정렬**되어 있어야 함
  - 최소: 1 (첫 번째 pivot 위치의 원소가 탐색 대상일 경우)
  - 최대: $\log_2 N$ (마지막 레벨의 pivot 원소가 탐색 대상일 경우)

◆ **예: 1,000,000개의 원소를 가진 배열에서 지정된 값 탐색**

- 순차탐색: 평균 500,000회 비교
- 이진탐색: 최대 20회 ($\because (\log_2 10^6) = 20$) 비교

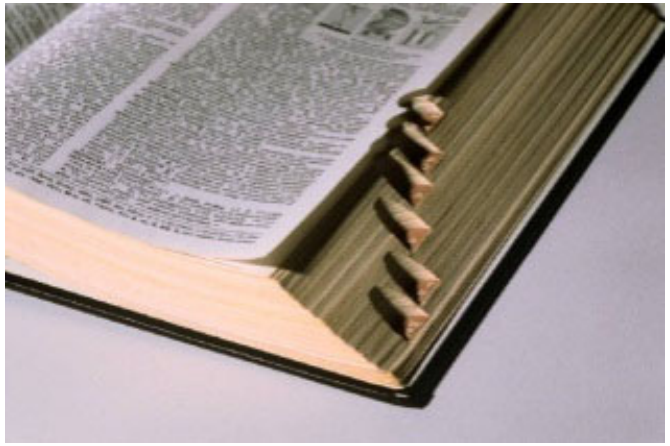
선택 정렬 (Selection Sorting)

정렬 (sorting) 이란?

- 정렬은 물건을 크기순으로 오름차순이나 내림차순으로 나열하는 것
- 정렬은 컴퓨터 공학분야에서 가장 기본적이고 중요한 알고리즘중의 하나

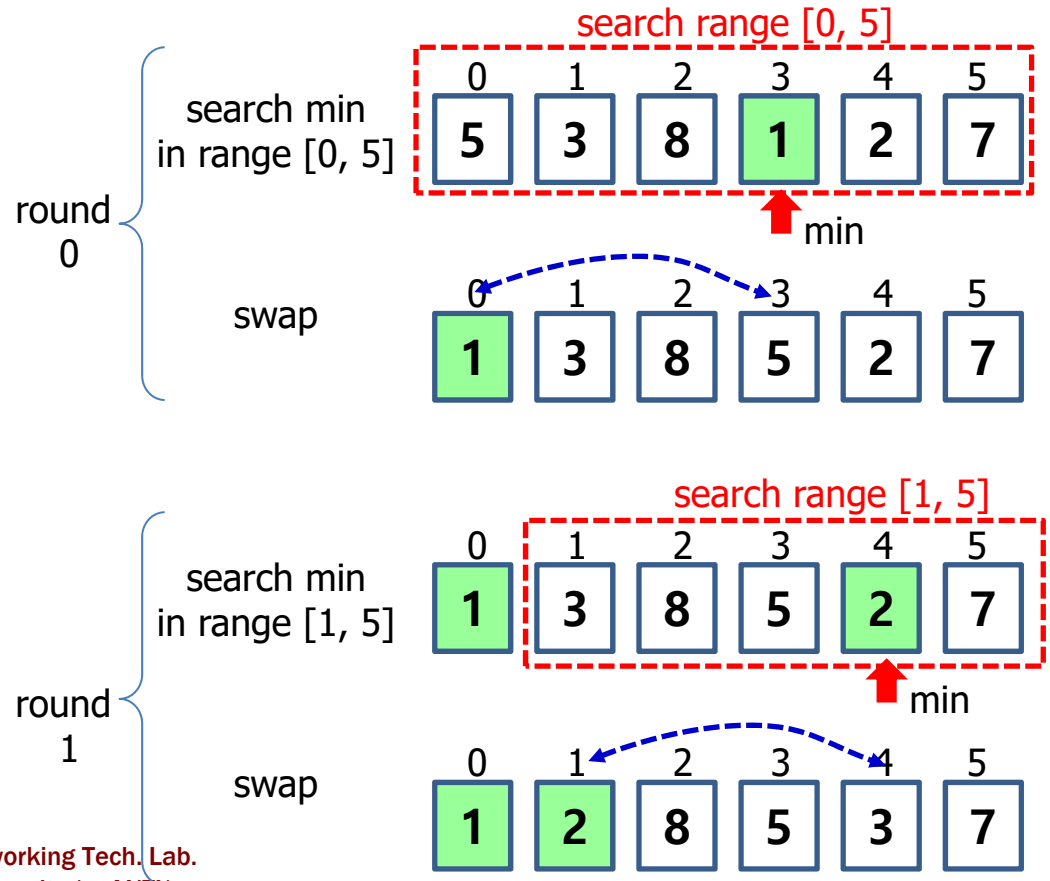
정렬이란?

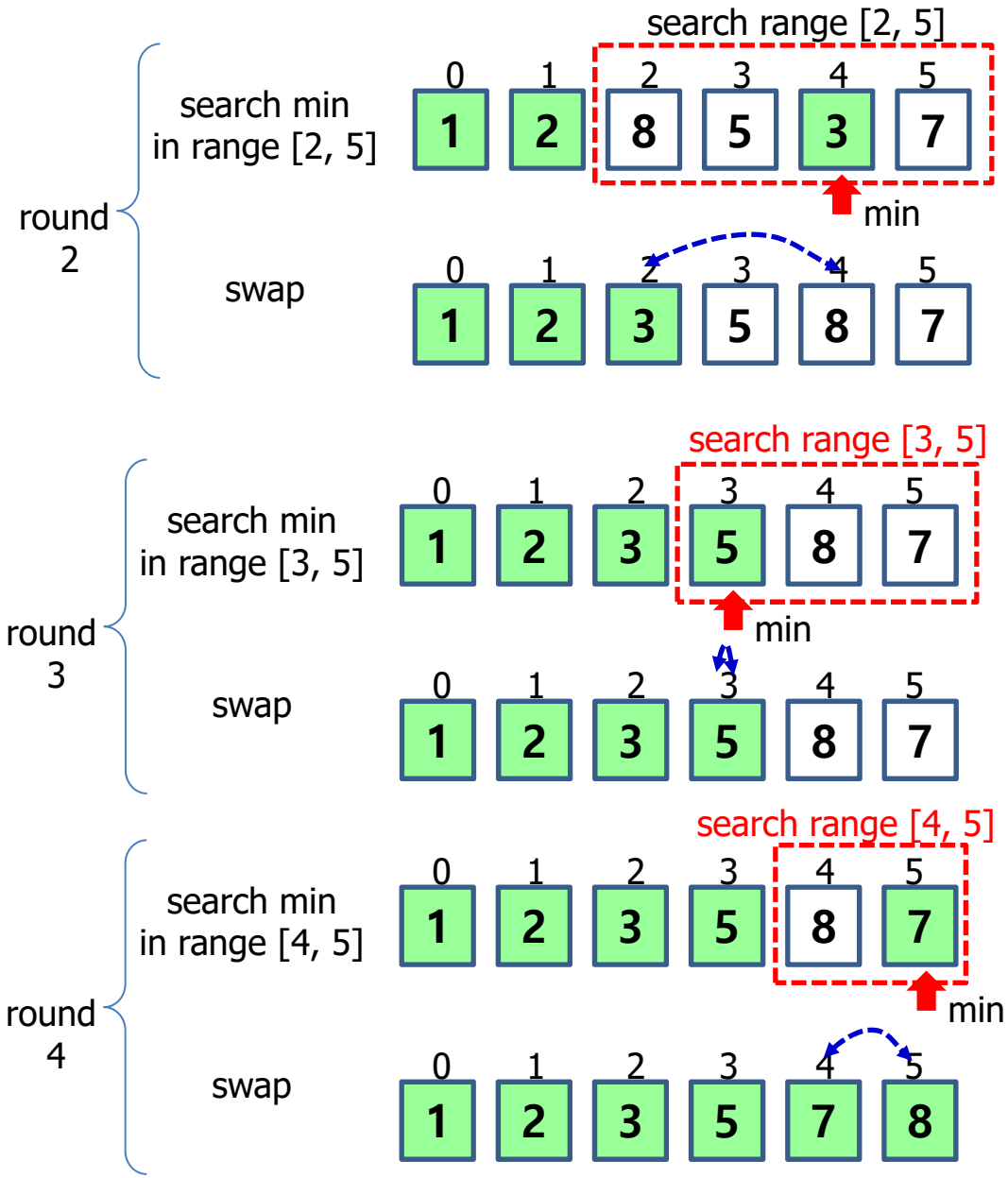
◆ 정렬은 자료 탐색에 있어서 필수적이다.
(예) 만약 사전에서 단어들이 정렬이 안되어 있다면?



선택정렬(selection sort)

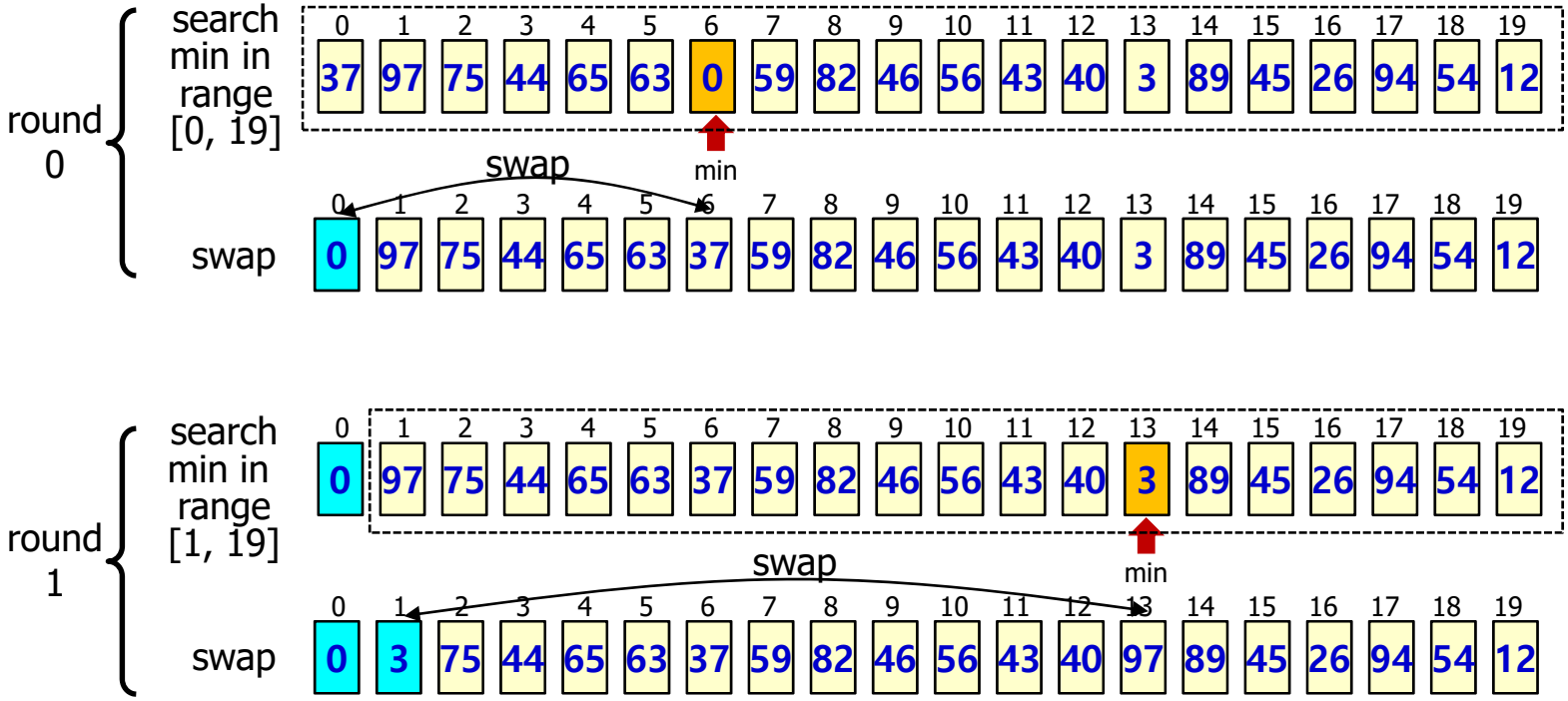
- 선택정렬(selection sort): 정렬이 안된 숫자들 중에서 최소값을 선택하여 배열의 첫 번째 요소와 교환
- 몇 개의 단계만 살펴보자.

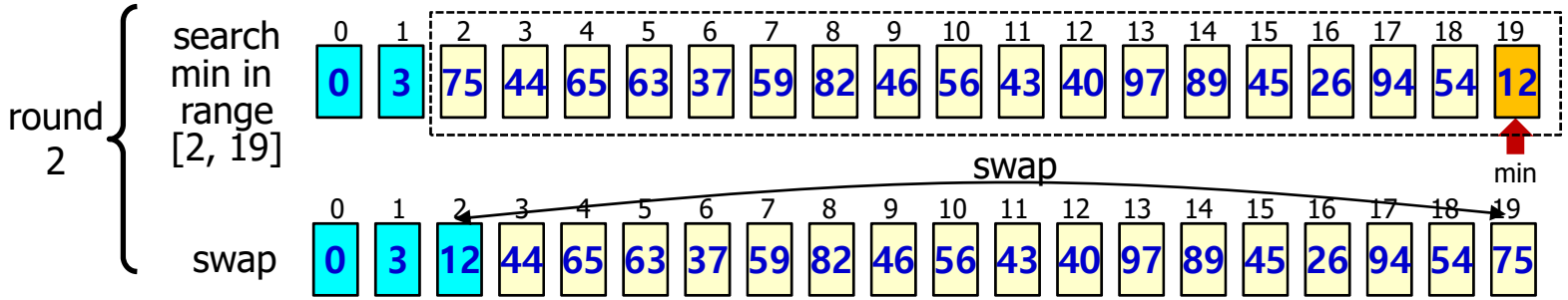



search range [2, 5]
round 2
search min in range [2, 5]
0 1 2 3 4 5
1 2 8 5 3 7
min
swap
0 1 2 3 4 5
1 2 3 5 8 7
search range [3, 5]
round 3
search min in range [3, 5]
0 1 2 3 4 5
1 2 3 5 8 7
min
swap
0 1 2 3 4 5
1 2 3 5 8 7
search range [4, 5]
round 4
search min in range [4, 5]
0 1 2 3 4 5
1 2 3 5 8 7
min
swap
0 1 2 3 4 5
1 2 3 5 7 8

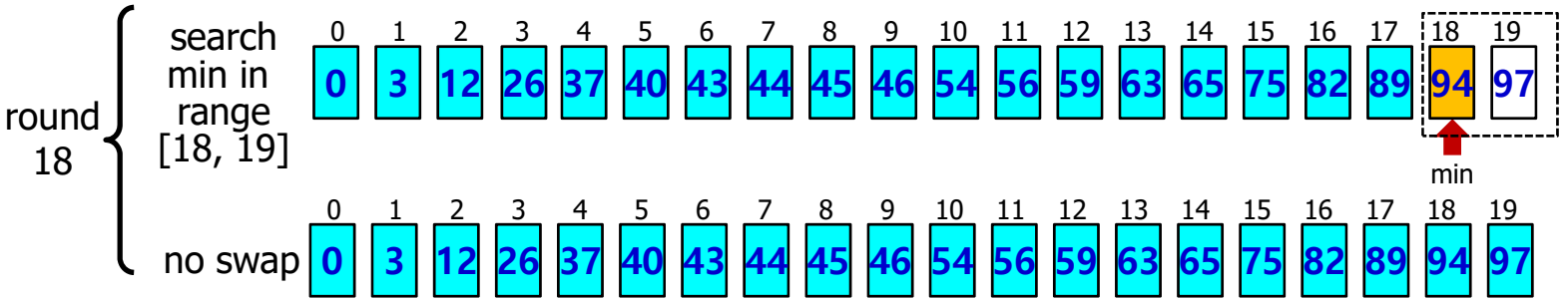
선택정렬(selection sorting)

◆ 선택정렬(selection sort): 정렬이 안된 숫자들 중에서 최소값을 선택하여 배열의 첫 번째 요소와 교환




round 2
search min in range [2, 19]
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19
0 3 75 44 65 63 37 59 82 46 56 43 40 97 89 45 26 94 54 12
min
swap
swap
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19
0 3 12 44 65 63 37 59 82 46 56 43 40 97 89 45 26 94 54 75

round 3 ~ 17 • • • •


round 18
search min in range [18, 19]
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19
0 3 12 26 37 40 43 44 45 46 54 56 59 63 65 75 82 89 94 97
min
no swap
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19
0 3 12 26 37 40 43 44 45 46 54 56 59 63 65 75 82 89 94 97

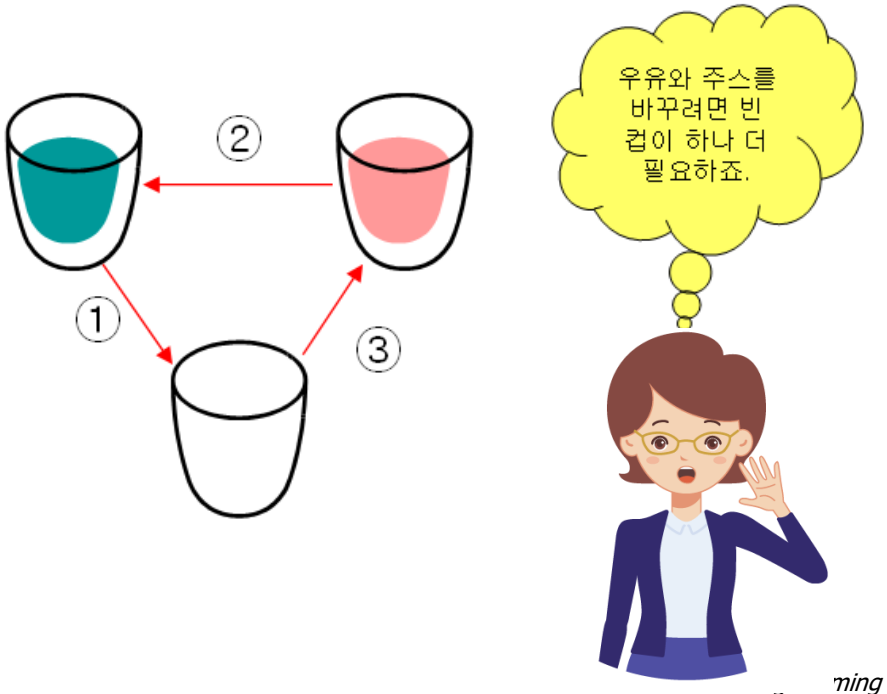
변수의 값을 서로 교환 (swap) 할 때

◆ 다음과 같이 하면 틀림

- grade[i] = grade[least]; // grade[i]의 기존값이 파괴된다!
- grade[least] = grade[i];

◆ 올바른 방법

- temp = list[i];
- list[i] = list[least];
- list[least] = temp;



선택 정렬 (Selection Sorting)

```
void selectionSort(int *array, int size)
{
	int index_min;  // index of minimum element
	int minElement; // value of the minimum element

	for (int round = 0; round < size - 1; round++)
	{
#ifdef DEBUG_SELECTION_SORT
		printf("Search range[%2d %2d]",  round, size - 1);
#endif
		index_min = round;
		minElement = array[round];
		for (int j = round + 1; j < size; j++)  //find the minimum element in array[round] ~ array[size-1]
		{
			if (array[j] < minElement)
			{
				index_min = j;
				minElement = array[j];
			}
		}
		if (index_min != round) /* if smaller element was found, then swap */
		{
			/* swap array[index_min] and array[round] minElement already has array[index_min] */
			array[index_min] = array[round];
			array[round] = minElement;
		}
#ifdef DEBUG_SELECTION_SORT
		printf("after swapping in round (%2d): ",  round);
		printArray(array, size, 20);
#endif
	}
}
```

선택정렬 응용 프로그램

```
void testSelectionSort(FILE *fout)
{
      int data_array[NUM_DATA] =
         { 37, 97, 75, 44, 65, 63, 0, 59, 82, 46, 56, 43, 40, 3, 89, 45, 26, 94, 54, 12 };
      printf("Integer array before selection_sorting:\n");
      printArray(data_array, NUM_DATA, LINE_SIZE);
      selectionSort(data_array, NUM_DATA);
      printf("Integer array after selection_sorting:\n");
      printArray(data_array, NUM_DATA, LINE_SIZE);
}
```

선택 정렬 실행 결과

```
Test Array Algorithms :
   1: Test Sequential Search
   2: Test Binary Search
   3: Test Selection Sort
   4: Test Quick Sort
   5: Test Selection Sort for Words
   6: Test Quick Sort for Words
   7: Performance Measurements of Quick Sort for Integer Array
   8: Performance Measurements of Selection Sort for Integer Array
   Esc: terminate
Input menu :
Integer array before selection_sorting:
   37   97   75   44   65   63    0   59   82   46   56   43   40    3   89   45   26   94   54   12
Search range[ 0 19]after swapping in round ( 0):    0   97   75   44   65   63   37   59   82   46   56   43   40    3   89   45   26   94   54   12
Search range[ 1 19]after swapping in round ( 1):    0    3   75   44   65   63   37   59   82   46   56   43   40   97   89   45   26   94   54   12
Search range[ 2 19]after swapping in round ( 2):    0    3   12   44   65   63   37   59   82   46   56   43   40   97   89   45   26   94   54   75
Search range[ 3 19]after swapping in round ( 3):    0    3   12   26   65   63   37   59   82   46   56   43   40   97   89   45   44   94   54   75
Search range[ 4 19]after swapping in round ( 4):    0    3   12   26   37   63   65   59   82   46   56   43   40   97   89   45   44   94   54   75
Search range[ 5 19]after swapping in round ( 5):    0    3   12   26   37   40   65   59   82   46   56   43   63   97   89   45   44   94   54   75
Search range[ 6 19]after swapping in round ( 6):    0    3   12   26   37   40   43   59   82   46   56   65   63   97   89   45   44   94   54   75
Search range[ 7 19]after swapping in round ( 7):    0    3   12   26   37   40   43   44   82   46   56   65   63   97   89   45   59   94   54   75
Search range[ 8 19]after swapping in round ( 8):    0    3   12   26   37   40   43   44   45   46   56   65   63   97   89   82   59   94   54   75
Search range[ 9 19]after swapping in round ( 9):    0    3   12   26   37   40   43   44   45   46   56   65   63   97   89   82   59   94   54   75
Search range[10 19]after swapping in round (10):    0    3   12   26   37   40   43   44   45   46   54   65   63   97   89   82   59   94   56   75
Search range[11 19]after swapping in round (11):    0    3   12   26   37   40   43   44   45   46   54   56   63   97   89   82   59   94   65   75
Search range[12 19]after swapping in round (12):    0    3   12   26   37   40   43   44   45   46   54   56   59   97   89   82   63   94   65   75
Search range[13 19]after swapping in round (13):    0    3   12   26   37   40   43   44   45   46   54   56   59   63   89   82   97   94   65   75
Search range[14 19]after swapping in round (14):    0    3   12   26   37   40   43   44   45   46   54   56   59   63   65   82   97   94   89   75
Search range[15 19]after swapping in round (15):    0    3   12   26   37   40   43   44   45   46   54   56   59   63   65   75   97   94   89   82
Search range[16 19]after swapping in round (16):    0    3   12   26   37   40   43   44   45   46   54   56   59   63   65   75   82   94   89   97
Search range[17 19]after swapping in round (17):    0    3   12   26   37   40   43   44   45   46   54   56   59   63   65   75   82   89   94   97
Search range[18 19]after swapping in round (18):    0    3   12   26   37   40   43   44   45   46   54   56   59   63   65   75   82   89   94   97
Integer array after selection_sorting:
   0    3   12   26   37   40   43   44   45   46   54   56   59   63   65   75   82   89   94   97
```

퀵 정렬 (Quick Sorting)

퀵정렬(Quick Sorting)

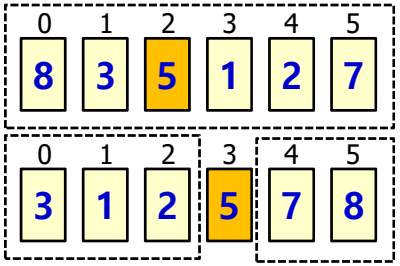
◆ 퀵 정렬 (Quick Sorting) 알고리즘

- 분할 및 정복 (divide and conquer) 방식의 알고리즘
- 탐색 구간의 중간 값을 pivot으로 선정하고, partition() 함수에서 이 pivot 보다 작은 원소들의 집합과 큰 원소들의 집합으로 분할 (pivot 원소의 위치는 변경될 수 있음)
- 분할된 각 구간에 대하여 quick_sort() 함수를 재귀함수 호출 (recursive function call)
- partition() 함수에서 pivot과의 비교 기능과 swapping 기능 수행
- partition() 함수를 사용한 분할 기능으로 탐색 구간을 ½ 정도씩으로 줄여감
- quick_sort() 함수의 재귀함수 호출에서 함수 호출의 오버헤드가 발생하며, 따라서 배열의 원소 개수가 작을 경우 선택정렬(selection sorting) 보다 늦은 성능을 가질 수 있음

Pivot Index, Pivot Vector와 Partition

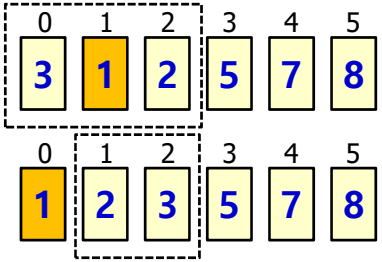
QS(A[0, 5])
pi = 2
pv = 5

Partition
new_pi = 3



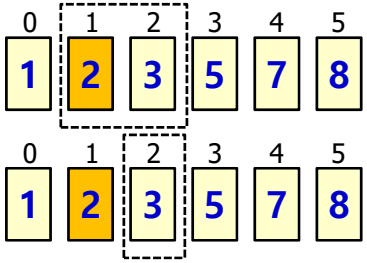
QS(A[0, 2])
pi = 1
pv = 1

Partition
new_pi = 0



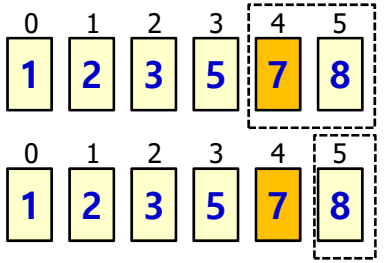
QS(A[0, 2])
pi = 1
pv = 2

Partition
new_pi = 1



QS(A[4, 5])
pi = 4
pv = 7

Partition
new_pi = 4



_partition()

```
int _partition(int *array, int size, int left, int right, int pivotIndex)
{
	int pivotValue; // pivot value
	int newPI; // new pivot index
	int temp, i;

	pivotValue = array[pivotIndex];

	temp = array[pivotIndex];
	array[pivotIndex] = array[right];
	array[right] = temp; // Move pivot to end

	newPI = left;
	for (i = left; i <= (right - 1); i++) {
		if (array[i] <= pivotValue) {
			temp = array[i];
			array[i] = array[newPI];
			array[newPI] = temp;
			newPI = newPI + 1;
		}
	}

	// swap array[newPI] and array[right]; Move pivot to its final place
	temp = array[newPI];
	array[newPI] = array[right];
	array[right] = temp;

	return newPI;
}
```

Processing in Partition()

QS(A[0, 19])
pi = 9
pv = 46

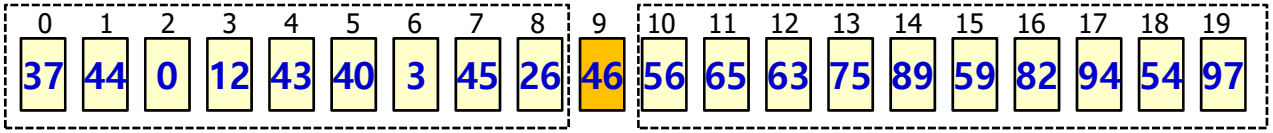
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
37	97	75	44	65	63	0	59	82	46	56	43	40	3	89	45	26	94	54	12

i	nPI	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
		37	97	75	44	65	63	0	59	82	46	56	43	40	3	89	45	26	94	54	12
0	1	37	97	75	44	65	63	0	59	82	12	56	43	40	3	89	45	26	94	54	46
1	1	37	97	75	44	65	63	0	59	82	12	56	43	40	3	89	45	26	94	54	46
2	1	37	97	75	44	65	63	0	59	82	12	56	43	40	3	89	45	26	94	54	46
3	2	37	44	75	97	65	63	0	59	82	12	56	43	40	3	89	45	26	94	54	46
4	2	37	44	75	97	65	63	0	59	82	12	56	43	40	3	89	45	26	94	54	46
5	2	37	44	75	97	65	63	0	59	82	12	56	43	40	3	89	45	26	94	54	46
6	3	37	44	0	97	65	63	75	59	82	12	56	43	40	3	89	45	26	94	54	46
7	3	37	44	0	97	65	63	75	59	82	12	56	43	40	3	89	45	26	94	54	46
8	3	37	44	0	97	65	63	75	59	82	12	56	43	40	3	89	45	26	94	54	46
9	4	37	44	0	12	65	63	75	59	82	97	56	43	40	3	89	45	26	94	54	46

Processing in Partition() (cont.)

i	nPl	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
9	4	37	44	0	12	65	63	75	59	82	97	56	43	40	3	89	45	26	94	54	46
10	4	37	44	75	12	65	63	75	59	82	97	56	43	40	3	89	45	26	94	54	46
11	5	37	44	75	12	43	63	75	59	82	97	56	65	40	3	89	45	26	94	54	46
12	6	37	44	75	12	43	40	75	59	82	97	56	65	63	3	89	45	26	94	54	46
13	7	37	44	75	12	43	40	3	59	82	97	56	65	63	75	89	45	26	94	54	46
14	7	37	44	75	12	43	40	3	59	82	97	56	65	63	75	89	45	26	94	54	46
15	8	37	44	75	12	43	40	3	45	82	97	56	65	63	75	89	59	26	94	54	46
16	9	37	44	75	12	43	40	3	45	26	97	56	65	63	75	89	59	82	94	54	46
17	9	37	44	75	12	43	40	3	45	26	97	56	65	63	75	89	59	82	94	54	46
18	9	37	44	75	12	43	40	3	45	26	97	56	65	63	75	89	59	82	94	54	46
		37	44	75	12	43	40	3	45	26	46	56	65	63	75	89	59	82	94	54	97

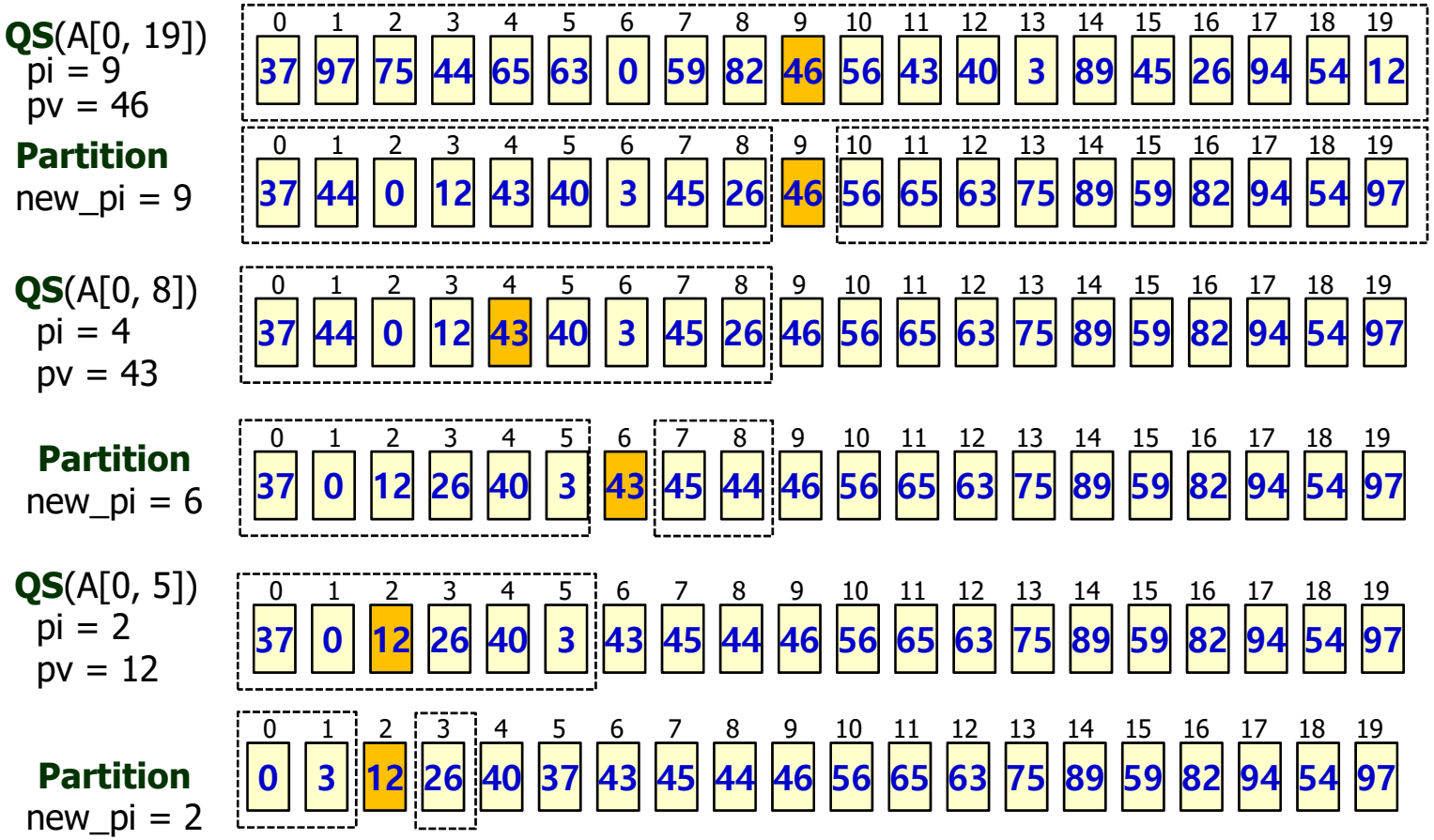
Partition
new_pi = 9



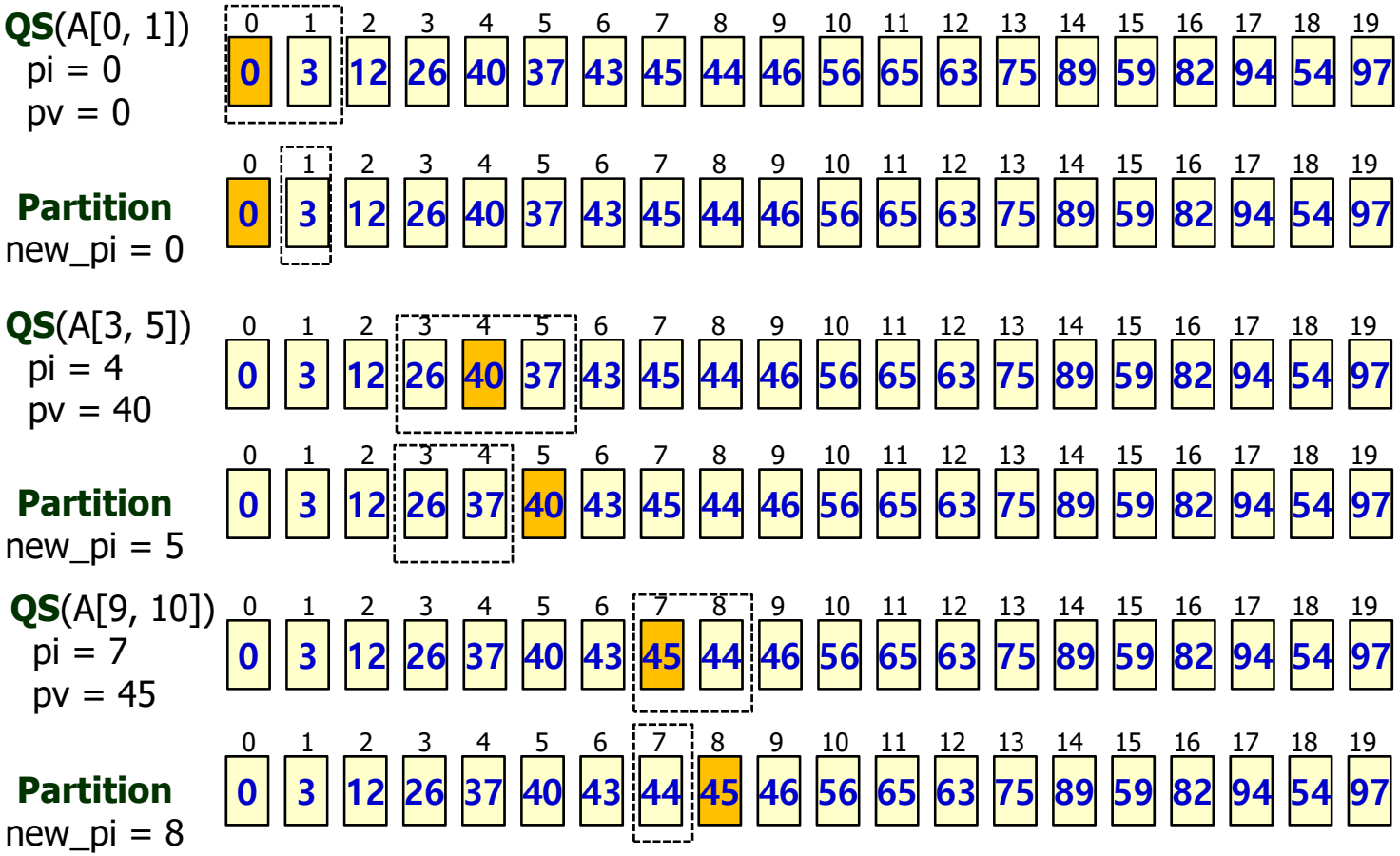
quickSort()

```
void _quickSort(int *array, int size, int left, int right)
{
	int pI, newPI; // pivot index

	if (left >= right) {
		return;
	}
	else if (left < right) { // subarray of 0 or 1 elements already sorted
		//select a pI (pivotIndex) in the range left ≤ pI ≤ right
		pI = (left + right) / 2;
	}
	newPI = _partition(array, size, left, right, pI);
	// element at newPivotIndex (newPI) is now at its final position

	if (left < (newPI - 1)) {
		_quickSort(array, size, left, newPI - 1);
		// recursively sort elements on the left of pivotNewIndex
	}
	if ((newPI + 1) < right) {
		_quickSort(array, size, newPI + 1, right);
		// recursively sort elements on the right of pivotNewIndex
	}
}

void quickSort(int *array, int size)
{
	_quickSort(array, size, 0, size - 1);
}
```

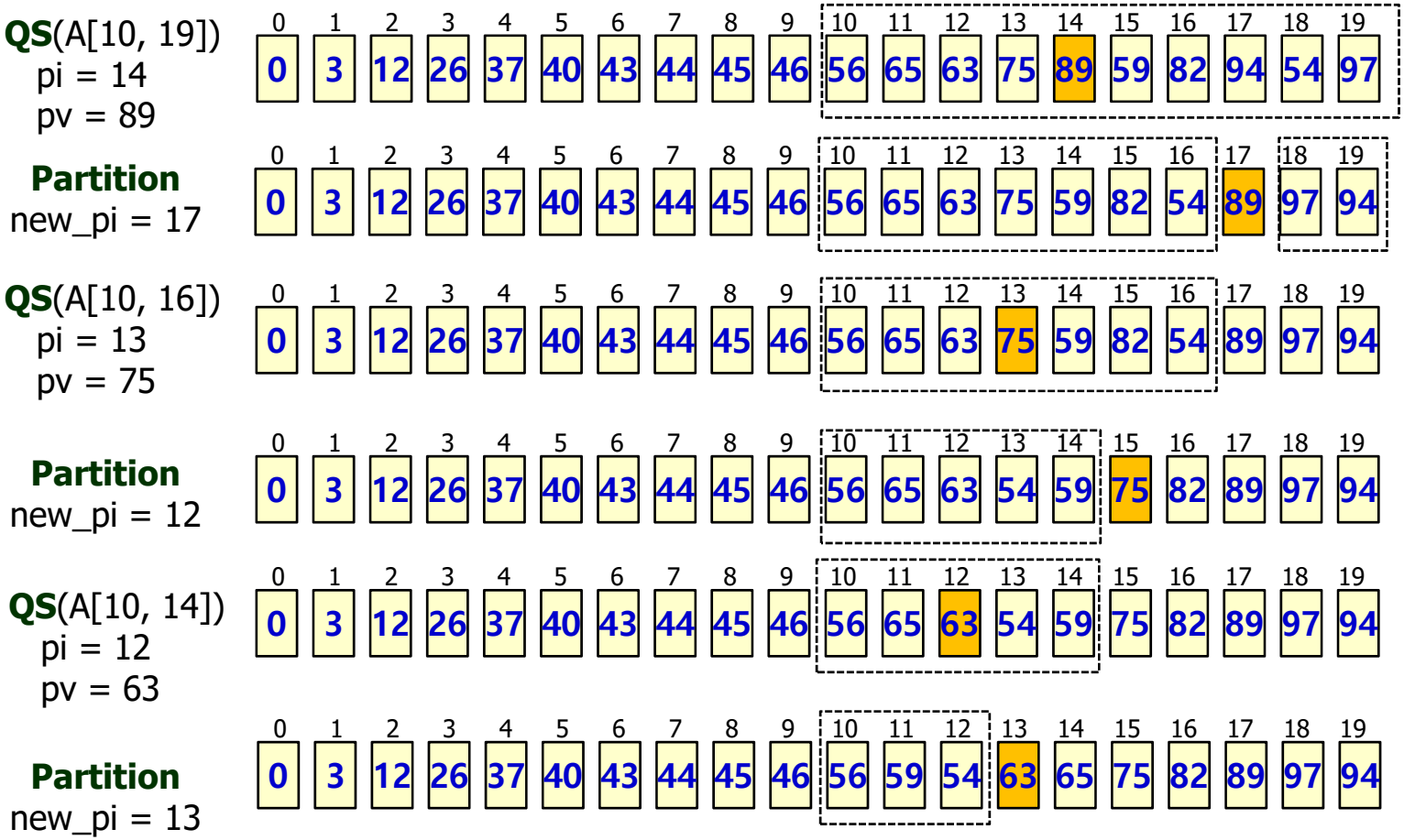
Quick Sorting (1)



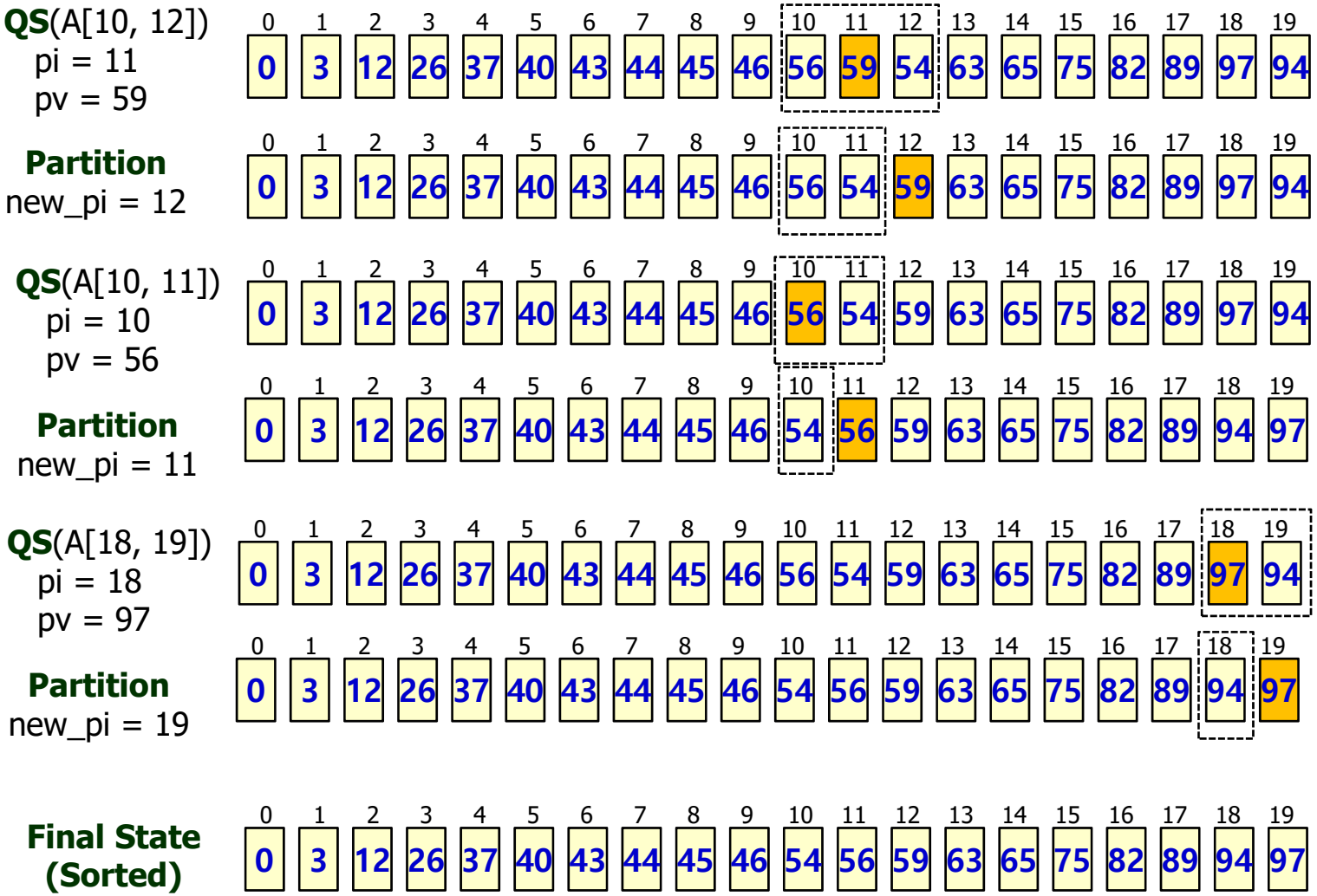
Quick Sorting (2)



Quick Sorting (3)



Quick Sorting (4)



main()

```
void testQuickSort(FILE *fout)
{
      int data_array[NUM_DATA] =
         { 37, 97, 75, 44, 65, 63, 0, 59, 82, 46, 56, 43, 40, 3, 89, 45, 26, 94, 54, 12 };

      printf("Integer array before quickSorting:\n");
      printArray(data_array, NUM_DATA, LINE_SIZE);
      quickSort(data_array, NUM_DATA);
      printf("Integer array after quickSorting:\n");
      printArray(data_array, NUM_DATA, LINE_SIZE);
}
```

퀵 정렬 실행 결과

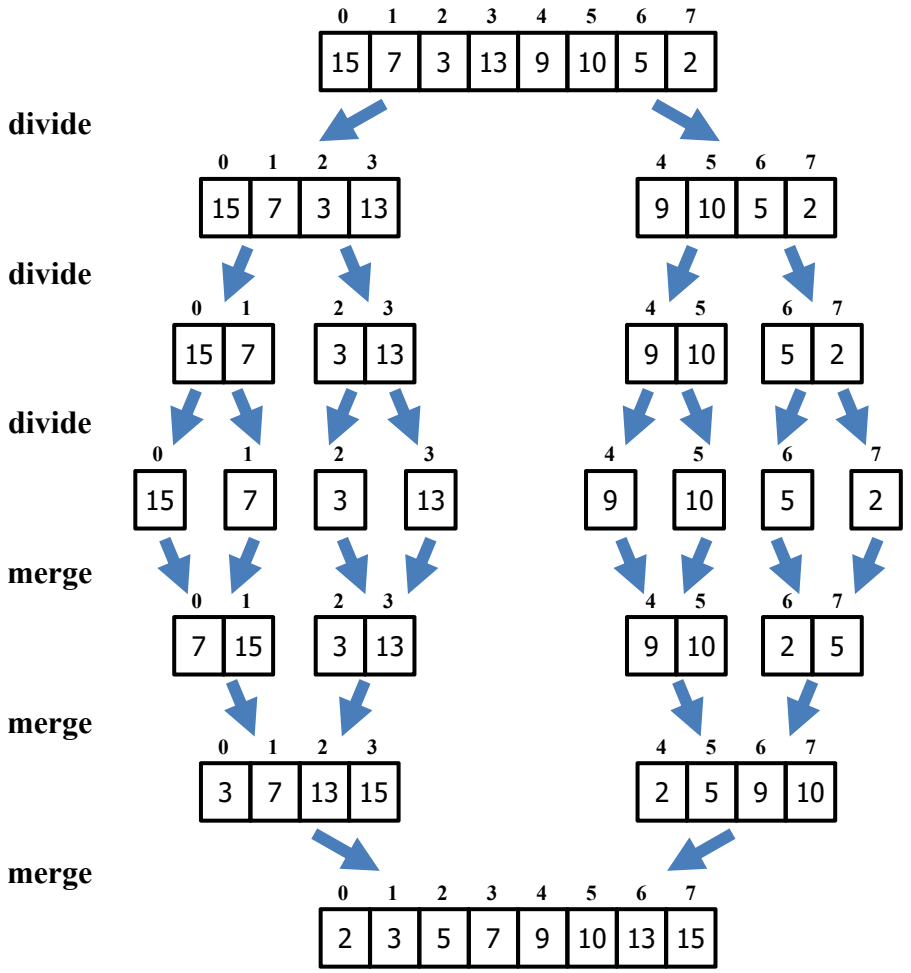
```
Integer array before quick_sorting:
  37   97   75   44   65   63    0   59   82   46   56   43   40    3   89   45   26   94   54   12
QuickSort (level:  0 [ 0 .. 19])
Partition : left ( 0) right (19) pivotIdex ( 9) pivotValue (46) =>  newPivotIndex ( 9) newPivotValue (46)
  37 44  0 12 43 40  3 45 26 <46>  56 65 63 75 89 59 82 94 54 97
   QuickSort (level:  1 [ 0 ..  8])
   Partition : left ( 0) right ( 8) pivotIdex ( 4) pivotValue (43) =>  newPivotIndex ( 6) newPivotValue (43)
    37  0 12 26 40  3 <43>  45 44
     QuickSort (level:  2 [ 0 ..  5])
     Partition : left ( 0) right ( 5) pivotIdex ( 2) pivotValue (12) =>  newPivotIndex ( 2) newPivotValue (12)
      0  3 <12>  26 40 37
       QuickSort (level:  3 [ 0 ..  1])
       Partition : left ( 0) right ( 1) pivotIdex ( 0) pivotValue ( 0) =>  newPivotIndex ( 0) newPivotValue ( 0)
      < 0>  3
       QuickSort (level:  3 [ 3 ..  5])
       Partition : left ( 3) right ( 5) pivotIdex ( 4) pivotValue (40) =>  newPivotIndex ( 5) newPivotValue (40)
      26 37 <40>
          QuickSort (level:  4 [ 3 ..  4])
          Partition : left ( 3) right ( 4) pivotIdex ( 3) pivotValue (26) =>  newPivotIndex ( 3) newPivotValue (26)
        <26>  37
     QuickSort (level:  2 [ 7 ..  8])
     Partition : left ( 7) right ( 8) pivotIdex ( 7) pivotValue (45) =>  newPivotIndex ( 8) newPivotValue (45)
     44 <45>
   QuickSort (level:  1 [10 .. 19])
   Partition : left (10) right (19) pivotIdex (14) pivotValue (89) =>  newPivotIndex (17) newPivotValue (89)
    56 65 63 75 59 82 54 <89>  97 94
     QuickSort (level:  2 [10 .. 16])
     Partition : left (10) right (16) pivotIdex (13) pivotValue (75) =>  newPivotIndex (15) newPivotValue (75)
     56 65 63 54 59 <75>  82
        QuickSort (level:  3 [10 .. 14])
        Partition : left (10) right (14) pivotIdex (12) pivotValue (63) =>  newPivotIndex (13) newPivotValue (63)
       56 59 54 <63>  65
          QuickSort (level:  4 [10 .. 12])
          Partition : left (10) right (12) pivotIdex (11) pivotValue (59) =>  newPivotIndex (12) newPivotValue (59)
         56 54 <59>
             QuickSort (level:  5 [10 .. 11])
             Partition : left (10) right (11) pivotIdex (10) pivotValue (56) =>  newPivotIndex (11) newPivotValue (56)
          54 <56>
     QuickSort (level:  2 [18 .. 19])
     Partition : left (18) right (19) pivotIdex (18) pivotValue (97) =>  newPivotIndex (19) newPivotValue (97)
     94 <97>
Integer array after quick_sorting:
   0    3   12   26   37   40   43   44   45   46   54   56   59   63   65   75   82   89   94   97
```

병합 정렬 (Merge Sorting)

병합 정렬 (Merge Sort)

◆ 병합정렬 알고리즘

- 전체 알고리즘이 분할 (divide)과 병합(merge) 단계로 구성됨
- 분할 단계에서는 주어진 정렬 구간을 반복적으로 ½로 나누고, 최종적으로 2개씩의 원소가 남으면 이를 정렬
- 병합 단계에서는 정렬된 부분 구간들을 병합시키면서 정렬 기능을 수행
- 병합단계에서는 이미 부분 구간들이 정렬되어 있으므로 병합정렬이 신속하게 처리될 수 있음



merge_sort()

```
void _mergeSort(int *array, int *tmp_array, int left, int right)
{
        printf("... invoke MergeSort(left=%d, right=%d):\n", left, right);

        if (left >= right)
                return;
        int i, j, k, mid = (left + right) / 2;
        _mergeSort(array, tmp_array, left, mid);
        _mergeSort(array, tmp_array, mid + 1, right);

        /* merge step 1 : copy 2nd half to tmp_array[] */
        for (i = mid; i >= left; i--)
                tmp_array[i] = array[i];
        for (j = 1; j <= right - mid; j++)
                tmp_array[right - j + 1] = array[j + mid];
        /* merge step 2 : merge sub-arrays back to array[] */
        for (i = left, j = right, k = left; k <= right; k++)
        {
                if (tmp_array[i] < tmp_array[j])
                        array[k] = tmp_array[i++];
                else
                        array[k] = tmp_array[j--];
        }
```

merge_sort()

```
        /* for debugging of merge_sort() */
#ifdef DEBUG_MERGESORT
        printf("After merging (left=%d, mid=%d, right=%d):\n", left, mid, right);
        printf("tempA: ");
        for (int i = left; i <= right; i++)
                printf("%5d", tmp_array[i]);
        printf("\n");
        printf(" A: ");
        for (int i = left; i <= right; i++)
                printf("%5d", array[i]);
        printf("\n");
#endif
}

void mergeSort(int *array, int size)
{
        int* tmp_array = (int*)calloc(size, sizeof(int));
        if (tmp_array == NULL)
        {
                printf("Error in creation of tmp_array (size = %d) in mergeSort() !!!\n");
                exit;
        }
        _mergeSort(array, tmp_array, 0, size - 1);
}
```

testMergeSort()

```
void testMergeSort(FILE* fout)
{
        int data_array[NUM_DATA] = { 37, 97, 75, 44, 65, 63, 0, 59, 82, 46,
            56, 43, 40, 3, 89, 45, 26, 94, 54, 12 };

        printf("Integer array before merge_sorting:\n");
        printArray(data_array, NUM_DATA, LINE_SIZE);
        mergeSort(data_array, NUM_DATA);
        printf("Integer array after merge_sorting:\n");
        printArray(data_array, NUM_DATA, LINE_SIZE);
}
```

```
Integer array before merge_sorting:
  37   97   75   44   65   63    0   59   82   46   56   43   40    3   89   45   26   94   54   12
Integer array after merge_sorting:
   0    3   12   26   37   40   43   44   45   46   54   56   59   63   65   75   82   89   94   97
```

정렬 알고리즘의 비교

◆ Selection sorting

- 2중 for-loop으로 구성
- inner for-loop에서는 각 구간에서의 최소값을 탐색
- outer for-loop에서는 inner for-loop에서 탐색된 최소값을 해당 구간의 시작 지점에 저장 (이를 위하여 swap 기능 수행)
- outer for-loop의 index: i = 0 ~ N-2
- inner for-loop의 index: j = (i + 1) ~ N-1

◆ Quick sorting

- 분할 및 정복 (divide and conquer) 방식의 알고리즘
- 탐색 구간의 중간 값을 pivot으로 선정하고, _partition() 함수에서 이 pivot 보다 작은 원소들의 집합과 큰 원소들의 집합으로 분할 (pivot 원소의 위치는 변경될 수 있음)
- 분할된 각 구간에 대하여 _quickSort() 함수를 재귀함수 호출 (recursive function call)
- _partition() 함수에서 pivot과의 비교 기능과 swapping 기능 수행
- _partition() 함수를 사용한 분할 기능으로 탐색 구간을 ½ 정도씩으로 줄여감
- _quickSort() 함수의 재귀함수 호출에서 함수 호출의 오버헤드가 발생하며, 따라서 배열의 원소 개수가 작을 경우 선택정렬(selection sorting) 보다 늦은 성능을 가질 수 있음

문자열 검색 (Search)과 정렬 (Sorting)

문자열(string) 관련 알고리즘 구현

```
/* String_Algorithms.h */
#ifndef STRING_ALGORITHMS_H
#define STRING_ALGORITHMS_H

#include <stdio.h>
#include <string.h>
#define MAX_WORD_LEN 15
#define MIN_WORD_LEN 5
#define MAX_NUM_WORDS 500
#define NUM_ALPHABET 26
#define WORDS_PER_LINE 10
#define SAMPLE_LINES 5

int genBigRand(int range);
void genWord(char word[], int min_word_len, int max_word_len);
void genRandWordArray(char wordArray[][MAX_WORD_LEN], int size);
void printWords(char wordArray[][MAX_WORD_LEN], int size);
void printBigWordArray(char wordArray[][MAX_WORD_LEN], int num_words, int words_per_line, int
    sample_lines);
void fprintBigWordArray(FILE *fout, char wordList[][MAX_WORD_LEN], int num_words, int words_per_line,
    int sample_lines);
int countWords(char wordArray[][MAX_WORD_LEN], int size);
void suffleWordArray(char wordArray[][MAX_WORD_LEN], int size);
void selectionSortWordArray(char wordArray[][MAX_WORD_LEN], int size);
int sequentialSearchWord(char wordArray[][MAX_WORD_LEN], int size, char key[]);
int binarySearchWord(char wordArray[][MAX_WORD_LEN], int size, char key[]);
int partitionWordArray(char wordArray[][MAX_WORD_LEN], int size, int left, int right, int
    pivotIndex, int level);
void quickSortWordArray(char words[][MAX_WORD_LEN], int size);
void _quickSortWordArray(char wordArray[][MAX_WORD_LEN], int size, int left, int right, int
    level);

#endif
```

문자열(string)의 비교와 Swap

◆ 문자열의 비교 (comparison)

- strcmp(char s1[], char s2[]) 함수를 사용

```
if (strcmp(words[i], words[j]) > 0)
{
      // words[i] is greater than words[j]
}
```

◆ 문자열의 swap

- strcpy(char s1[], char s2[]) 함수를 사용

```
char temp[MAX_WORD_LENGTH+1];

strcpy(temp, words[i]);
strcpy(words[i], words[j]);
strcpy(words[j], temp);
```

문자열 검색 - searchWord()

```
int sequentialSearchWord(char wordList[][MAX_WORD_LEN], int size, char key[])
{
        for (int pos = 0; pos < size; pos++)
        {
                if (strcmp(wordList[pos], "-1") == 0)
                        return -1;
                if (strcmp(wordList[pos], key) == 0)
                        return pos;
        }
        return -1;
}
```

문자열(string)의 선택정렬 (sorting)

```
void selectionSortWordArray(char wordList[][MAX_WORD_LEN],   int size)
{
        char temp [MAX_WORD_LEN];
        int min_pos;

        for (int pos = 0; pos < size; pos++)
        {
                min_pos = pos;
                for (int j = pos + 1; j < size; j++)
                {
                        if (strcmp(wordList[min_pos], wordList[j]) > 0)
                        {
                                min_pos = j;
                        }
                }
                if (min_pos != pos)
                {
                        strcpy(temp, wordList[pos]);
                        strcpy(wordList[pos], wordList[min_pos]);
                        strcpy(wordList[min_pos], temp);
                }
        }
}
```

문자열(string)의 선택정렬 응용 프로그램

```
void testSelectionSort_Words(FILE *fout)
{
    char word_array[MAX_NUM_WORDS][MAX_WORD_LEN] = { "" };
    char temp[MAX_WORD_LEN] = { '\0' };
    FILE *fin;
    const char *input_file_name = "Word_List.txt";
    int word_count = 0;

    fin = fopen(input_file_name, "r");
    if (fin == NULL)
    {
        printf("Error in opening word_list (%s)\n", input_file_name);
        return;
    }
    word_count = 0;
    while (fscanf(fin, "%s", temp) != EOF)
    {
        //printf("%2d-th input word: %s\n", word_count, temp);
        strcpy(word_array[word_count], temp);
        word_count++;;
    }
```

```
    printf("Word list before selection sorting:\n");
    printWords(word_array, word_count);
    selectionSortWordArray(word_array, word_count);
    printf("Word list after selection sorting of word list :\n");
    printWords(word_array, word_count);

    printf("Shuffling word list . . . \n");
    suffleWordArray(word_array, word_count);

    printf("Word list after shuffling:\n");
    printWords(word_array, word_count);
    selectionSortWordArray(word_array, word_count);
    printf("Word list after selection sorting of word list :\n");
    printWords(word_array, word_count);
    fclose(fin);
}
```

문자열 배열의 정렬 기능 시험을 위한 입력 파일

◆ Word_List.txt

double char auto default const continue break case do
union unsigned void volatile while
return short signed sizeof static struct switch typeof
else enum extern float for goto if int long register

문자열 배열의 정렬 - 실행결과

```
Test Array Algorithms :
   1: Test Sequential Search
   2: Test Binary Search
   3: Test Selection Sort
   4: Test Quick Sort
   5: Test Selection Sort for Words
   6: Test Quick Sort for Words
   7: Performance Measurements of Quick Sort for Integer Array
   8: Performance Measurements of Selection Sort for Integer Array
   Esc: terminate
Input menu :
Word list before selection sorting:
double        char          auto          default       const         continue      break         case          do            union
unsigned      void          volatile      while         return        short         signed        sizeof        static        struct
switch        typeof        else          enum          extern        float         for           goto          if            int
long          register
Word list after selection sorting of word list :
auto          break         case          char          const         continue      default       do            double        else
enum          extern        float         for           goto          if            int           long          register      return
short         signed        sizeof        static        struct        switch        typeof        union         unsigned      void
volatile      while
Shuffling word list . . .
Word list after shuffling:
extern        if            else          auto          struct        static        case          char          register      while
sizeof        typeof        enum          int           goto          default       do            long          switch        return
break         union         continue      for           short         void          double        volatile      unsigned      const
float         signed
Word list after selection sorting of word list :
auto          break         case          char          const         continue      default       do            double        else
enum          extern        float         for           goto          if            int           long          register      return
short         signed        sizeof        static        struct        switch        typeof        union         unsigned      void
volatile      while
```

프로그램 모듈의 실행시간 정밀측정

프로그램 모듈 (함수)의 실행 시간 측정

◆ 프로그램 모듈(함수)의 실행 시간 측정

- 모듈(함수)의 실행 직전과 실행 직후의 시간을 각각 측정하고, 그 경과시간을 계산
- time() 함수: 초단위 경과시간 측정
- Windows 운영체제의 performance counter: 밀리초/마이크로초 단위의 정밀한 경과시간 측정

◆ Windows 운영체제 환경에서의 Performance Counter 기반 경과시간 측정

- CPU 클럭 (2GHz 이상)을 기반으로 구동되는 performance counter를 사용하여 milli-second/micro-second 단위의 경과시간을 정밀하게 측정
- 사용되는 Windows 시스템 라이브러리 함수
  - ***QueryPerformanceFrequency(LARGE_INTEGER &freq);***
  - ***QueryPerformanceCounter (LARGE_INTEGER & time);***

Performance Counter in Windows

◆ MS-Windows에서 사용되는 추가적인 자료형

- LARGE_INTEGER: union type with 64-bit integer (QuadPart)
- LONGLONG: 64-bit integer

◆ Performance Frequency

- ticks-per-second

◆ Performance Counter

- CPU clock ticks 단위로 계수
- 2GHz 이상의 CPU clock => nano (10^{-9})-second 단위 측정가능

◆ Query Performance Counter

- Retrieves the current value of the performance counter (ticks), which is a high resolution (<1us) time stamp that can be used for time-interval measurements.

(Source: https://www.pluralsight.com/blog/software-development/how-to-measure-execution-time-intervals-in-c--)

◆ Sample Test Driver

```
#include <Windows.h>
. . . .
int function_to_be_tested(int array[], int size);
// QueryPerformanceFrequency(LARGER_INTEGER *freq);
// QueryPerformanceCounter(LARGER_INTEGER *time);

int main()
{
   LARGE_INTEGER  freq, t_before, t_after;
   LONGLONG  t_diff;
   double  elapsed_time;
   int *array, size, base = 0;

   printf("input size of big array = ");
   scanf("%d", &size);
   array = (int *) calloc(size, sizeof(int)); // 동적배열 생성
   genBigRandArray(array, size, base);

  QueryPerformanceFrequency(&freq);  // CPU clock frequency의 기록

  QueryPerformanceCounter(&t_before); // 함수 실행 직전 시간 기록
  selectionSort (array, size);
  QueryPerformanceCounter(&t_after); // 함수 실행 직후 시간 기록

   t_diff = t_after.QuadPart - t_before.QuadPart;
   elapsed_time = ((double) t_diff / freq.QuadPart);  // in second

   printf("It took %10.3lf [milliseconds] to perform the function with %d
            integer data array.", elapsed_time*1000, size);
   . . . .
}
```

정렬 알고리즘의 실행 시간 측정 및 비교 (1)

```
void PM_QuickSort_IntArray(FILE *fout)
{
	int *big_int_array;
	LARGE_INTEGER freq, t_before, t_after;
	LONGLONG t_diff;
	double elapsed_time;

	QueryPerformanceFrequency(&freq);
	srand(0);
	for (int array_size = 100000; array_size <= 500000; array_size += 100000)
	{
		big_int_array = (int *)calloc(array_size, sizeof(int));
		if (big_int_array == NULL)
		{
			printf("Error in memory allocation for big_int_array of size (%d) !!!\n", array_size);
			return;
		}
		genBigRandArray(big_int_array, array_size, 0);
		fprintf(fout, "Big integer array before quick sorting : \n");
		fprintBigArraySample(fout, big_int_array, array_size, 20, 2);
		printf("Quick sorting of an integer array (size : %7d) . . . . ", array_size);
		QueryPerformanceCounter(&t_before);
		quickSort(big_int_array, array_size);
		QueryPerformanceCounter(&t_after);
		fprintf(fout, "Big integer array after quick sorting : \n");
		fprintBigArraySample(fout, big_int_array, array_size, 20, 2);
		t_diff = t_after.QuadPart – t_before.QuadPart;
		elapsed_time = (double)t_diff / freq.QuadPart;
		fprintf(fout, "Quick sorting of an integer array (size : %7d) took %10.2lf [milliseconds]\n",
		    array_size, elapsed_time * 1000.0);
		printf(" took %10.2lf [milliseconds]\n", elapsed_time * 1000.0);
	}
}
```

```
void PM_SelectionSort_IntArray(FILE *fout)
{
	int *big_int_array;
	LARGE_INTEGER freq, t_before, t_after;
	LONGLONG t_diff;
	double elapsed_time;

	QueryPerformanceFrequency(&freq);
	srand(0);
	for (int array_size = 100000; array_size <= 500000; array_size += 100000)
	{
		big_int_array = (int *)calloc(array_size, sizeof(int));
		if (big_int_array == NULL)
		{
			printf("Error in memory allocation for big_int_array of size (%d) !!!\n", array_size);
			return;
		}
		genBigRandArray(big_int_array, array_size, 0);
		fprintf(fout, "\nBig integer array before selection sorting : \n");
		fprintBigArraySample(fout, big_int_array, array_size, 20, 2);
		printf("Selection sorting of an integer array (size : %7d) . . . .  ", array_size);
		QueryPerformanceCounter(& t_before);
		selectionSort(big_int_array, array_size);
		QueryPerformanceCounter(& t_after);
		fprintf(fout, "Big integer array after selection sorting : \n");
		fprintBigArraySample(fout, big_int_array, array_size, 20, 2);
		t_diff = t_after.QuadPart - t_before.QuadPart;
		elapsed_time = (double)t_diff / freq.QuadPart;
		fprintf(fout, "Selection sorting of an integer array (size : %7d) took %10.2lf [ms]\n",
		  array_size, elapsed_time * 1000.0);
		printf(" took %10.2lf [ms]\n", elapsed_time * 1000.0);
	}
}
```

정렬 알고리즘의 실행 시간 측정 및 비교 (화면 출력)

```
Test Array Algorithms :
   1: Test Sequential Search
   2: Test Binary Search
   3: Test Selection Sort
   4: Test Quick Sort
   5: Test Selection Sort for Words
   6: Test Quick Sort for Words
   7: Performance Measurements of Quick Sort for Integer Array
   8: Performance Measurements of Selection Sort for Integer Array
   Esc: terminate
Input menu : 7
Quick sorting of an integer array (size :  100000) . . . .  took      13.59 [milliseconds]
Quick sorting of an integer array (size :  200000) . . . .  took      28.53 [milliseconds]
Quick sorting of an integer array (size :  300000) . . . .  took      43.14 [milliseconds]
Quick sorting of an integer array (size :  400000) . . . .  took      58.57 [milliseconds]
Quick sorting of an integer array (size :  500000) . . . .  took      73.75 [milliseconds]

Test Array Algorithms :
   1: Test Sequential Search
   2: Test Binary Search
   3: Test Selection Sort
   4: Test Quick Sort
   5: Test Selection Sort for Words
   6: Test Quick Sort for Words
   7: Performance Measurements of Quick Sort for Integer Array
   8: Performance Measurements of Selection Sort for Integer Array
   Esc: terminate
Input menu : 8
Selection sorting of an integer array (size :  100000) . . . .   took   10088.15 [milliseconds]
Selection sorting of an integer array (size :  200000) . . . .   took   40341.00 [milliseconds]
Selection sorting of an integer array (size :  300000) . . . .   took   90744.53 [milliseconds]
Selection sorting of an integer array (size :  400000) . . . .   took  161324.25 [milliseconds]
Selection sorting of an integer array (size :  500000) . . . .   took  252103.05 [milliseconds]
```

정렬 알고리즘의 실행 시간 측정 및 비교 (파일 출력)

```
Big integer array before quick sorting :
   52903    29221    72437    36605    56212    19204    21337    72337    36045    20376    73459    16946      299    74848     1078    51118    43707    29813    97865    59075
   44222    29151    91080    93927    32970    16476    71777    57633     3848     1472    45688    77806     1140     2672    60036    43709    69247    55622    41071    62932
    . . . . .
   35273     2706     6119    91355    12996    30504     5717    48035    90027    76471    63567    65208    89079    84208    15780    20839    41884     2558     4687    44275
   38576    38268    70043    21926    75263    79430    70719    82413    65825    82536    22054     1246    37154    63028     2222    15046    64022    10164    71878    66104

Big integer array after quick sorting :
       0        1        2        3        4        5        6        7        8        9       10       11       12       13       14       15       16       17       18       19
      20       21       22       23       24       25       26       27       28       29       30       31       32       33       34       35       36       37       38       39
    . . . . .
   99960    99961    99962    99963    99964    99965    99966    99967    99968    99969    99970    99971    99972    99973    99974    99975    99976    99977    99978    99979
   99980    99981    99982    99983    99984    99985    99986    99987    99988    99989    99990    99991    99992    99993    99994    99995    99996    99997    99998    99999

Quick sorting of an integer array (size :  100000) took      13.59 [milliseconds]
Big integer array before quick sorting :
    2605    44075   139154    61476   155268    48461    98926   158965    38708   106388    97182    94857   152825    33893    45306   123181   180645    63900   146349   124840
  192669    57606   179702   114928   190938   165862   148328   137395    76841   189809    61803    66831     6123   173010   157688    95898    62294   187623    33953    97115
    . . . . .
  161627   195866   144599    39603   146108    50170     5858   182275   129972    25063   109421    87123    51095    22706   107416    24593   146329   179592   168688   162668
  156841   103658    94645   127267   136756   140140   152309   198751   107092   138498   165204   106089   183323    22737     8358    63453   135103    16117    29059   179139

Big integer array after quick sorting :
       0        1        2        3        4        5        6        7        8        9       10       11       12       13       14       15       16       17       18       19
      20       21       22       23       24       25       26       27       28       29       30       31       32       33       34       35       36       37       38       39
    . . . . .
  199960   199961   199962   199963   199964   199965   199966   199967   199968   199969   199970   199971   199972   199973   199974   199975   199976   199977   199978   199979
  199980   199981   199982   199983   199984   199985   199986   199987   199988   199989   199990   199991   199992   199993   199994   199995   199996   199997   199998   199999

Quick sorting of an integer array (size :  200000) took      28.53 [milliseconds]


Selection sorting of an integer array (size :  300000) took   90744.53 [milliseconds]

Big integer array before selection sorting :
   18289   255510   325747   111530   277387   129217   369693   174336    13059   243131   267220     5211   327014   212483   366481   134643    46646   374188   186240   287188
   28010   311773   145154   183394   158700   334929   372521    53455   103962   201176   223167   355622   124229   158805   324288   395076   177973   304830   374235    85978
    . . . . .
   74001     6076   175984    23574   272251    60855   125817   384258   207437   317838   232519   274645    35364   376641   392391   238011   193781    97708   148896     7510
  138616    36880   361414    69474   134406   147225   302553   276570   230558    60833   164503   131705    80176   253106   371961   334937   120976   333342   314603   224819

Big integer array after selection sorting :
       0        1        2        3        4        5        6        7        8        9       10       11       12       13       14       15       16       17       18       19
      20       21       22       23       24       25       26       27       28       29       30       31       32       33       34       35       36       37       38       39
    . . . . .
  399960   399961   399962   399963   399964   399965   399966   399967   399968   399969   399970   399971   399972   399973   399974   399975   399976   399977   399978   399979
  399980   399981   399982   399983   399984   399985   399986   399987   399988   399989   399990   399991   399992   399993   399994   399995   399996   399997   399998   399999

Selection sorting of an integer array (size :  400000) took  161324.25 [milliseconds]

Big integer array before selection sorting :
  227287   316862   195603   218424   425007   210879   472588   478115   398309    83792   406429   163757    52346   313220   123154   266275   187109    93155   289515    70151
  204208    83078   336508   255629   429208   209247   310542   222910   278480   155119   171740   219960   414431   430061     2775   368524    28253   323686   140067   120863
    . . . . .
  228817   403535   333598   479799   116372    20655    10087   100679   250158   466264   495947   147804   462762   292129   292413   106243   331733   333170   406473   388058
  311355   445415   334374     3693   340601   373614   147928   129819    26056   326592   157837   261657   496771   277608   455036   264570   362347   482784   454573   135015

Big integer array after selection sorting :
       0        1        2        3        4        5        6        7        8        9       10       11       12       13       14       15       16       17       18       19
      20       21       22       23       24       25       26       27       28       29       30       31       32       33       34       35       36       37       38       39
    . . . . .
  499960   499961   499962   499963   499964   499965   499966   499967   499968   499969   499970   499971   499972   499973   499974   499975   499976   499977   499978   499979
  499980   499981   499982   499983   499984   499985   499986   499987   499988   499989   499990   499991   499992   499993   499994   499995   499996   499997   499998   499999

Selection sorting of an integer array (size :  500000) took  252103.05 [milliseconds]
```

Performance Comparisons for Quick Sort and Selection Sort for Small Arrays

```
void PM_Comp_Sorting_SmallIntArray(FILE *fout)  /* (1) */
{
      int *int_array;
      LARGE_INTEGER freq, t_before, t_after;
      LONGLONG t_diff;
      double elapsed_time;

      QueryPerformanceFrequency(&freq);
      srand(0);
      for (int array_size = 5; array_size <= 200; array_size += 5)
      {
            int_array = (int *)calloc(array_size, sizeof(int));
            if (int_array == NULL)
            {
                  printf("Error in memory allocation for big_int_array of size (%d) !!!\n",
                     array_size);
                  return;
            }
            genBigRandArray(int_array, array_size, 0);
            fprintf(fout, "Big integer array before quick sorting : \n");
            fprintBigArraySample(fout, int_array, array_size, 20, 2);
            printf("Sorting of an integer array (size : %3d) : ", array_size);
            QueryPerformanceCounter(& t_before);
            quickSort(int_array, array_size);
            QueryPerformanceCounter(& t_after);
```

```
/* void PM_Comp_Sorting_SmallIntArray(FILE *fout) (2) */

        fprintf(fout, "Big integer array after quick sorting : \n");
        fprintBigArraySample(fout, int_array, array_size, 20, 2);
        t_diff = t_after.QuadPart - t_before.QuadPart;
        elapsed_time = (double)t_diff / freq.QuadPart;
        fprintf(fout, "Quick sorting of an integer array (size : %7d) took %10.2lf [us]\n",
           array_size, elapsed_time * 1000000.0);
        printf(" QuickSort took %6.2lf [micro-seconds], ", elapsed_time * 1000000.0;

        fprintf(fout, "Shuffling word list . . . \n");
        suffleArray(int_array, array_size);
        fprintf(fout, "Big integer array before quick sorting : \n");
        fprintBigArraySample(fout, int_array, array_size, 20, 2);
        //printf("Quick sorting of an integer array (size : %7d) . . . . ", array_size);
        QueryPerformanceCounter(&t_before);
        selectionSort(int_array, array_size);
        QueryPerformanceCounter(& t_after);
        fprintf(fout, "Big integer array after quick sorting : \n");
        fprintBigArraySample(fout, int_array, array_size, 20, 2);
        t_diff = t_after.QuadPart - t_before.QuadPart;
        elapsed_time = (double)t_diff / freq.QuadPart;
        fprintf(fout, "Quick sorting of an integer array (size : %7d) took %10.2lf [us]\n",
           array_size, elapsed_time * 1000000.0);
        printf("  Selection Sorting took %6.2lf [micro-seconds]\n",
           elapsed_time * 1000000.0);
        free(int_array);
    }
}
```

```
Sorting of an integer array (size :   5) :  QuickSort took   0.51 [micro-seconds],   Selection Sorting took   0.26 [micro-seconds]
Sorting of an integer array (size :  10) :  QuickSort took   0.51 [micro-seconds],   Selection Sorting took   0.51 [micro-seconds]
Sorting of an integer array (size :  15) :  QuickSort took   1.03 [micro-seconds],   Selection Sorting took   0.51 [micro-seconds]
Sorting of an integer array (size :  20) :  QuickSort took   1.28 [micro-seconds],   Selection Sorting took   1.03 [micro-seconds]
Sorting of an integer array (size :  25) :  QuickSort took   1.80 [micro-seconds],   Selection Sorting took   1.28 [micro-seconds]
Sorting of an integer array (size :  30) :  QuickSort took   2.05 [micro-seconds],   Selection Sorting took   1.80 [micro-seconds]
Sorting of an integer array (size :  35) :  QuickSort took   2.31 [micro-seconds],   Selection Sorting took   2.05 [micro-seconds]
Sorting of an integer array (size :  40) :  QuickSort took   2.82 [micro-seconds],   Selection Sorting took   2.57 [micro-seconds]
Sorting of an integer array (size :  45) :  QuickSort took   3.08 [micro-seconds],   Selection Sorting took   2.82 [micro-seconds]
Sorting of an integer array (size :  50) :  QuickSort took   3.34 [micro-seconds],   Selection Sorting took   3.59 [micro-seconds]
Sorting of an integer array (size :  55) :  QuickSort took   3.85 [micro-seconds],   Selection Sorting took   4.11 [micro-seconds]
Sorting of an integer array (size :  60) :  QuickSort took   4.11 [micro-seconds],   Selection Sorting took   5.13 [micro-seconds]
Sorting of an integer array (size :  65) :  QuickSort took   4.88 [micro-seconds],   Selection Sorting took   5.64 [micro-seconds]
Sorting of an integer array (size :  70) :  QuickSort took   5.64 [micro-seconds],   Selection Sorting took   6.67 [micro-seconds]
Sorting of an integer array (size :  75) :  QuickSort took   5.64 [micro-seconds],   Selection Sorting took   7.18 [micro-seconds]
Sorting of an integer array (size :  80) :  QuickSort took   6.16 [micro-seconds],   Selection Sorting took   7.95 [micro-seconds]
Sorting of an integer array (size :  85) :  QuickSort took   6.41 [micro-seconds],   Selection Sorting took   9.24 [micro-seconds]
Sorting of an integer array (size :  90) :  QuickSort took   6.67 [micro-seconds],   Selection Sorting took  10.01 [micro-seconds]
Sorting of an integer array (size :  95) :  QuickSort took   6.93 [micro-seconds],   Selection Sorting took  11.29 [micro-seconds]
Sorting of an integer array (size : 100) :  QuickSort took   7.70 [micro-seconds],   Selection Sorting took  12.06 [micro-seconds]
Sorting of an integer array (size : 105) :  QuickSort took   7.95 [micro-seconds],   Selection Sorting took  13.09 [micro-seconds]
Sorting of an integer array (size : 110) :  QuickSort took   8.98 [micro-seconds],   Selection Sorting took  14.37 [micro-seconds]
Sorting of an integer array (size : 115) :  QuickSort took   9.24 [micro-seconds],   Selection Sorting took  15.91 [micro-seconds]
Sorting of an integer array (size : 120) :  QuickSort took   9.24 [micro-seconds],   Selection Sorting took  16.93 [micro-seconds]
Sorting of an integer array (size : 125) :  QuickSort took   9.75 [micro-seconds],   Selection Sorting took  18.22 [micro-seconds]
Sorting of an integer array (size : 130) :  QuickSort took   9.75 [micro-seconds],   Selection Sorting took  19.76 [micro-seconds]
Sorting of an integer array (size : 135) :  QuickSort took  10.26 [micro-seconds],   Selection Sorting took  21.30 [micro-seconds]
Sorting of an integer array (size : 140) :  QuickSort took  11.29 [micro-seconds],   Selection Sorting took  24.12 [micro-seconds]
Sorting of an integer array (size : 145) :  QuickSort took  11.55 [micro-seconds],   Selection Sorting took  24.12 [micro-seconds]
Sorting of an integer array (size : 150) :  QuickSort took  12.06 [micro-seconds],   Selection Sorting took  25.92 [micro-seconds]
Sorting of an integer array (size : 155) :  QuickSort took  12.57 [micro-seconds],   Selection Sorting took  27.20 [micro-seconds]
Sorting of an integer array (size : 160) :  QuickSort took  12.57 [micro-seconds],   Selection Sorting took  28.99 [micro-seconds]
Sorting of an integer array (size : 165) :  QuickSort took  13.09 [micro-seconds],   Selection Sorting took  30.53 [micro-seconds]
Sorting of an integer array (size : 170) :  QuickSort took  13.09 [micro-seconds],   Selection Sorting took  32.59 [micro-seconds]
Sorting of an integer array (size : 175) :  QuickSort took  13.86 [micro-seconds],   Selection Sorting took  34.13 [micro-seconds]
Sorting of an integer array (size : 180) :  QuickSort took  14.63 [micro-seconds],   Selection Sorting took  36.18 [micro-seconds]
Sorting of an integer array (size : 185) :  QuickSort took  15.14 [micro-seconds],   Selection Sorting took  37.98 [micro-seconds]
Sorting of an integer array (size : 190) :  QuickSort took  15.40 [micro-seconds],   Selection Sorting took  40.28 [micro-seconds]
Sorting of an integer array (size : 195) :  QuickSort took  15.65 [micro-seconds],   Selection Sorting took  42.34 [micro-seconds]
Sorting of an integer array (size : 200) :  QuickSort took  15.91 [micro-seconds],   Selection Sorting took  44.13 [micro-seconds]
```

Homework 6

Homework 6

6.1 단어 (문자열) 배열의 선택정렬 및 퀵 정렬 구현, 실행시간 측정

1) rand() 함수를 사용하여 5 ~ 15자의 문자로 구성된 단어(문자열)을 생성하여 지정된 주소에 저장하는 함수 (void genWord(char word[], int min_word_len, int max_word_len)) 를 작성하라. 단어의 최소 길이 및 최대 길이는 min_word_len과 max_word_len으로 지정된다. 문자열의 첫 번째 문자는 대문자로 구성되어야 하며, 나머지 문자는 소문자로 구성된다. 문자의 마지막에는 반드시 '\0' (NULL character)이 포함되도록 할 것. 단어(문자열) 관련 함수들은 WordArray.cpp 파일에 구현하며, 이 함수들의 함수원형은 WordArray.h 헤더파일에 포함시킬 것.
2) 최대 MAX_WORD_LEN (15)자 길이의 문자열을 최대 MAX_NUM_WORDS (500,000)개 저장할 수 있는 2차원 문자 배열을 "WordArrayData.cpp" 소스파일에 선언하고, 위 1)에서 작성한 genWord() 함수를 사용하여 문자열을 생성하여 2차원 문자 배열의 값으로 초기화하라.
3) 큰 규모의 단어 배열의 첫부분과 마지막부분을 파일로 출력하는 함수 **void fprintBigWordArray(FILE *fout, char wordList[][MAX_WORD_LEN], int size, int words_per_line = 10, int sample_lines = 5)**를 작성하라. 이 함수는 한 줄에 words_per_line 개수의 단어를 출력하며, 첫 부분의 sample_lines, 마지막 부분의 sample_lines 수 만큼의 줄을 출력한다.
4) selection sorting 구조를 기반으로 NUM_WORDS 개의 문자열을 정렬하는 함수 void selectionSortWordArray(char word[][MAX_WORD_LEN], int num_words)를 작성하라. 정렬된 결과의 첫 번째 10개 단어 × 5줄과 맨 마지막 10개 단어 × 5줄을 fprintBigWordArray() 함수를 사용하여 파일로 출력하라.
5) quick sorting 구조를 기반으로 NUM_WORDS 개의 문자열을 정렬하는 함수 void quickSortWordArray(char word[][MAX_WORD_LEN], int num_words)를 작성하라. 정렬된 결과의 첫 번째 10개 단어 × 5줄과 맨 마지막 10개 단어 × 5줄을 fprintBigWordArray() 함수를 사용하여 파일로 출력하라.

6) Windows 운영체제에서 제공하는 Performance Counter를 사용하여 selectionSortWordArray() 함수와 quickSortWordArray() 함수가 10000 ~ 100000 개의 단어를 정렬할 때 걸린 시간을 millisecond 단위로 측정하여 출력하라. 10000개 단어씩 증가시키며 측정할 것.

7) 단어 배열의 정렬에서 selectionSortWordArray() 함수와 quickSortWordArray() 함수의 실행시간에 차이가 나는 이유를 설명하라. 만약 NUM_WORDS가 500,000 및 1,000,000이 되었을 때 어떤 차이가 나는지 예상하여 설명하라.

8) main() 함수

```
/* Performance Analysis for Sorting Algorithms of Word Array (1) */
#include <stdio.h>
#include <stdlib.h>
#include <Windows.h>
#include <time.h>
#include <string.h>
#include "WordArray.h"

void main()
{
	extern char words[][MAX_WORD_LEN];
	LARGE_INTEGER freq, t_before, t_after;
	LONGLONG t_diff;
	double elapsed_time;
	FILE *fout;

	fout = fopen("output.txt", "w");
	if (fout == NULL)
	{
		printf("Error in creation of output.txt file !!\n");
		exit(-1);
	}
	srand(time(NULL));
	QueryPerformanceFrequency(&freq);
```

```
/* Performance Analysis for Sorting Algorithms of Word Array (2) */

    printf("Performance Analysis of Sorting Algorithms\n");
    for (int num_words = 10000; num_words <= 100000; num_words += 10000)
    {
        printf("Word array(size: %7d) : ", num_words);
        fprintf(fout,
        "\n=============================================\n");
        genRandWordArray(words, num_words);
        fprintf(fout, "Randomly generated Word Array (size: %d)\n", num_words);
        fprintBigWordArray(fout, words, num_words, WORDS_PER_LINE, SAMPLE_LINES);

        fprintf(fout, "\nQuick sorting for Word Array (size: %d) .....\n", num_words);
        QueryPerformanceCounter(&t_before);
        quickSortWordArray(words, num_words, 0, num_words - 1, 0);
        QueryPerformanceCounter(&t_after);

        t_diff = t_after.QuadPart - t_before.QuadPart;
        elapsed_time = ((double)t_diff / freq.QuadPart); // in second
        fprintf(fout, "QuickSort_WordArray(size: %d) took %10.3f [millisecond]\n", num_words,
        elapsed_time *1000.0);
        printf("Quick_Sort (%10.3lf ms), ", elapsed_time *1000.0);

        fprintf(fout, "\nAfter sorting word array (size : %d)\n", num_words);
        fprintBigWordArray(fout, words, num_words, WORDS_PER_LINE, SAMPLE_LINES);
        fprintf(fout, "\n");
```

```
/* Performance Analysis for Sorting Algorithms of Word Array (3) */

        suffleWordArray(words, num_words);
        fprintf(fout, "Word Array (size: %d) after Suffling\n", num_words);
        fprintBigWordArray(fout, words, num_words, WORDS_PER_LINE, SAMPLE_LINES);

        fprintf(fout, "\nSelection sorting for Word Array (size: %d) .....\n", num_words);
        QueryPerformanceCounter(&t_before);
        selectionSortWordArray(words, num_words);
        QueryPerformanceCounter(&t_after);
        t_diff = t_after.QuadPart - t_before.QuadPart;
        elapsed_time = ((double)t_diff / freq.QuadPart); // in second
        fprintf(fout, "SelectionSort_WordArray(size: %d) took %10.3f [millisecond]\n",
        num_words, elapsed_time *1000.0);
        printf("Select_Sort (%10.3lf ms)\n", elapsed_time *1000.0);

        fprintf(fout, "\nAfter sorting word array (size : %d)\n", num_words);
        fprintBigWordArray(fout, words, num_words, WORDS_PER_LINE, SAMPLE_LINES);
        fprintf(fout, "\n");
    } // end for
    fclose(fout);
}
```

◆ 실행결과 (화면 출력)

```
Performance Analysis of Sorting Algorithms
Word array(size:      10) : Quick_Sort (     0.003 ms), Select_Sort (     0.001 ms)
Word array(size:      15) : Quick_Sort (     0.004 ms), Select_Sort (     0.002 ms)
Word array(size:      20) : Quick_Sort (     0.004 ms), Select_Sort (     0.003 ms)
Word array(size:      25) : Quick_Sort (     0.007 ms), Select_Sort (     0.003 ms)
Word array(size:      30) : Quick_Sort (     0.008 ms), Select_Sort (     0.004 ms)
Word array(size:      35) : Quick_Sort (     0.010 ms), Select_Sort (     0.005 ms)
Word array(size:      40) : Quick_Sort (     0.011 ms), Select_Sort (     0.006 ms)
Word array(size:      45) : Quick_Sort (     0.012 ms), Select_Sort (     0.023 ms)
Word array(size:      50) : Quick_Sort (     0.014 ms), Select_Sort (     0.009 ms)
Word array(size:      55) : Quick_Sort (     0.018 ms), Select_Sort (     0.010 ms)
Word array(size:      60) : Quick_Sort (     0.019 ms), Select_Sort (     0.012 ms)
Word array(size:      65) : Quick_Sort (     0.020 ms), Select_Sort (     0.014 ms)
Word array(size:      70) : Quick_Sort (     0.023 ms), Select_Sort (     0.017 ms)
Word array(size:      75) : Quick_Sort (     0.021 ms), Select_Sort (     0.018 ms)
Word array(size:      80) : Quick_Sort (     0.030 ms), Select_Sort (     0.020 ms)
Word array(size:      85) : Quick_Sort (     0.026 ms), Select_Sort (     0.021 ms)
Word array(size:      90) : Quick_Sort (     0.026 ms), Select_Sort (     0.024 ms)
Word array(size:      95) : Quick_Sort (     0.028 ms), Select_Sort (     0.026 ms)
Word array(size:     100) : Quick_Sort (     0.039 ms), Select_Sort (     0.028 ms)
Word array(size:     105) : Quick_Sort (     0.033 ms), Select_Sort (     0.032 ms)
Word array(size:     110) : Quick_Sort (     0.040 ms), Select_Sort (     0.033 ms)
Word array(size:     115) : Quick_Sort (     0.057 ms), Select_Sort (     0.036 ms)
Word array(size:     120) : Quick_Sort (     0.047 ms), Select_Sort (     0.039 ms)
Word array(size:     125) : Quick_Sort (     0.046 ms), Select_Sort (     0.042 ms)
Word array(size:     130) : Quick_Sort (     0.049 ms), Select_Sort (     0.045 ms)
Word array(size:     135) : Quick_Sort (     0.048 ms), Select_Sort (     0.048 ms)
Word array(size:     140) : Quick_Sort (     0.049 ms), Select_Sort (     0.050 ms)
Word array(size:     145) : Quick_Sort (     0.048 ms), Select_Sort (     0.056 ms)
Word array(size:     150) : Quick_Sort (     0.053 ms), Select_Sort (     0.057 ms)
Word array(size:     155) : Quick_Sort (     0.054 ms), Select_Sort (     0.061 ms)
Word array(size:     160) : Quick_Sort (     0.059 ms), Select_Sort (     0.064 ms)
Word array(size:     165) : Quick_Sort (     0.053 ms), Select_Sort (     0.067 ms)
Word array(size:     170) : Quick_Sort (     0.058 ms), Select_Sort (     0.075 ms)
Word array(size:     175) : Quick_Sort (     0.061 ms), Select_Sort (     0.074 ms)
Word array(size:     180) : Quick_Sort (     0.068 ms), Select_Sort (     0.079 ms)
Word array(size:     185) : Quick_Sort (     0.071 ms), Select_Sort (     0.087 ms)
Word array(size:     190) : Quick_Sort (     0.064 ms), Select_Sort (     0.087 ms)
Word array(size:     195) : Quick_Sort (     0.069 ms), Select_Sort (     0.090 ms)
Word array(size:     200) : Quick_Sort (     0.077 ms), Select_Sort (     0.095 ms)
계속하려면 아무 키나 누르십시오 . . .
```

```
Performance Analysis of Sorting Algorithms
Word array(size:   10000) : Quick_Sort (     5.733 ms), Select_Sort (   185.983 ms)
Word array(size:   20000) : Quick_Sort (    13.726 ms), Select_Sort (   742.444 ms)
Word array(size:   30000) : Quick_Sort (    19.839 ms), Select_Sort (  1669.898 ms)
Word array(size:   40000) : Quick_Sort (    26.645 ms), Select_Sort (  2970.281 ms)
Word array(size:   50000) : Quick_Sort (    36.134 ms), Select_Sort (  4640.933 ms)
Word array(size:   60000) : Quick_Sort (    41.910 ms), Select_Sort (  6677.167 ms)
Word array(size:   70000) : Quick_Sort (    48.561 ms), Select_Sort (  9089.018 ms)
Word array(size:   80000) : Quick_Sort (    63.451 ms), Select_Sort ( 11873.129 ms)
Word array(size:   90000) : Quick_Sort (    66.130 ms), Select_Sort ( 15016.963 ms)
Word array(size:  100000) : Quick_Sort (    77.896 ms), Select_Sort ( 18521.519 ms)
계속하려면 아무 키나 누르십시오 . . .
```

```
Performance Analysis of Sorting Algorithms
Word array(size:  100000) : Quick_Sort (    74.885 ms), Select_Sort ( 18565.403 ms)
Word array(size:  200000) : Quick_Sort (   155.070 ms), Select_Sort ( 74208.299 ms)
Word array(size:  300000) : Quick_Sort (   233.558 ms), Select_Sort (167068.997 ms)
Word array(size:  400000) : Quick_Sort (   341.565 ms), Select_Sort (297290.519 ms)
Word array(size:  500000) : Quick_Sort (   440.996 ms), Select_Sort (465546.442 ms)
계속하려면 아무 키나 누르십시오 . . .
```

◆ 실행결과 (파일 출력 일부분)

```
Randomly generated Word Array (size: 100000)
Sxnlfdeqc       Hzsfuexs        Rghpvwyzrvnp    Pmwlcazzfkuq    Mqhwugnvrm      Izmmmdld        Vjgkrnidnkm     Ijwijbzzkvaswi   Yjrxqhaccx      Tczhutc
Aajgmd          Gyejybgv        Zeshhb          Knekgswxzz      Zvtquxyhigleu   Bfwceaooyuyhk   Mntdmgkacttaoy  Cdjtyidjaeo      Uqwvjxg         Gbwziocfiwyt
Egxxdgrxplghxp  Hiyzgr          Zpyfoi          Ovzeaprowqchex  Meckwafuei      Qnckttohdtx     Yrzewa          Vienhkiccxoe     Ozoec           Yjagtjmupqcko
Drgqkmwsqkhoj   Wcplthdyaryjk   Mphcnnocenzzv   Tgmscsvpvu      Vuhpyqwcbg      Wlrrcgympb      Zjibtdgfzsgnc   Lpvaugwiiavxn    Qeusnz          Njmlcxffyjc
Dfipqye         Ebmmfiy         Ecmkgjjzqaalzw  Ldorlevautri    Wrkkmzpbhmf     Belprhpmb       Xnfnnrzjcctulf  Iheolp           Lfdjjxaaty      Lbdsnlqk
. . . .
Qddhohzx        Fmbvxfzxv       Itzcde          Xbmncrw         Friclcrhjjnam   Dzwphlzvp       Lndthb          Uvsapqripvbv     Jthcyluy        Pvexqv
Pzrvxmf         Gysewgesrpqabe  Rqozg           Nxvxez          Tsymrwtbumbd    Dggtk           Fipuaieizu      Kvnshffregwt     Ulmsqqhnknnt    Bnoqofvuw
Rduvsfq         Fbbsob          Jtpqlrmw        Yhdllfmh        Mbrwrhxsuw      Gaonlxpb        Beyslla         Kzcysehdrswf     Qbzbxouscj      Ynlcgipsehfjct
Mrzyci          Qaloyceznd g    Azhrdcva        Mtegmmy         Mlesarjpcsoi    Cktkbn          Ivlaqq          Xxdxggszparo     Enhfigkfbucdz   Tlpwyfs
Arxprpc         Xtuqcelmu       Zqjfevddg       Lcuggulpxait    Wtirvdrc        Aimuwixwbt      Kohpuof         Zricehpgqx       Jwbpjgq         Tbhpejzm

Quick sorting for Word Array (size: 100000) .....
QuickSort_WordArray(size: 100000) took     77.467 [millisecond]

After sorting word array (size : 100000)
Aaagj           Aaaminbprqqtl   Aaanjzatlqe     Aaare           Aaaxtltfunz     Aabdcpxzntm     Aabiov          Aabrpbxy         Aabvtshpg       Aaccbghvama
Aacckqsvvpw     Aachwozn        Aackcuy         Aacovtlmfpva    Aacqe           Aadajpejmggbam  Aadbiodkj       Aadcdygzqzf      Aadhrzqkzydm     Aadhyj
Aadpz           Aadwvwo         Aadwzqqtrpm     Aadyborn        Aaedqsdfzlblkx  Aaegwfampm      Aaejticfyycues  Aaenxtpphesg     Aaexex          Aafbkbnwwv
Aafdjzzbsbgqsj  Aafjhiwllzyvsr  Aafmwmxz        Aafnqsvwciy     Aagciuowisreha  Aagifm          Aagjogrvn       Aaguwxedqaj      Aahjpu          Aahkeeysgim
Aahnbijr        Aahvpczbfnp     Aahyytxskmkl    Aahzxipvfoyy    Aaibuut         Aaiegztg        Aaiizr          Aajckyvigjs      Aajeckcsuj      Aajeesrkqagzj
. . . .
Zzpjujvko       Zzpnsgickn      Zzprp           Zzqrdtlvyepfcb  Zzqwikmged      Zzqzra          Zzraprdlf       Zzrbct           Zzrevz          Zzrjzcghdin
Zzrkusea        Zzrmeof         Zzrybgi         Zzrzjviad       Zzsnvwuufh      Zzsqamkykk      Zzsqxaq         Zzstfagjpq       Zzsuzz          Zztafu
Zztyrkvsfopfkk  Zztzxhaqgukh    Zzuaem          Zzugctnfxwlqiw  Zzuilluwtnj     Zzujlgtn        Zzupfjyjvoibj   Zzuwehyo         Zzvagtcimi      Zzvcmbcruj
Zzvkculqzged    Zzvlaxdnqzs     Zzwbuwzmju      Zzwck           Zzwiwzrqcro     Zzwkxpzjmjd     Zzwlatkwmn      Zzwmgzbbkhpinm   Zzxjhh          Zzxki
Zzxkwgoya       Zzxoywsqp       Zzydwxha        Zzyjsmyolopb    Zzysmlfpdsf     Zzyznikixvx     Zzzfzr          Zzzjd            Zzzxewssfxmdvz   Zzzyyebmxtqlkv

Word Array (size: 100000) after Suffling
Pmfds           Itgzgqtqr       Djdxegxz        Mjvdxtlwvx      Ymxdupze        Eqzluzvydpdw    Slcecp          Rxbfz            Wrnrcfnpke      Mfyiqw
Aacckqsvvpw     Zcjfhs          Dkkrml          Hbiql           Aacqe           Xkuduauht       Qzxwjgphgjsba   Esrno            Lxtvwc          Aadhyj
Yzfdtfwcjlcydw  Newzzbjfg       Fudfc           Kiaijagujr      Otrpriehkotgfc  Uzvgeabynqtaoz  Dtbkfdbvcpx     Dckigjt          Coobuuysojsxb   Esiylbcbtphrgr
Aafdjzzbsbgqsj  Labwvrqdru      Tntwopfgrvbthi  Hsycjhfzk       Htyvkuwvfxlswp   Kuddvn          Nznvrlm         Ubvvyoxakzw      Lsttkos         Fikghwnuatfj
Lkzah           Heolmhpdjyj     Nptsxuswguvrkf  Txjbhsltosaxaw  Qzgquwoapp      Ccurflamlpfq    Doenamwjcrkdlu  Scgjmccrfwirp    Yluavqkr        Vjcxxsfbmny
. . . .
Ggqghixlhl      Uddnaqedlmk     Zhgjigtrnnv     Jznpzcectlqe    Zzqwikmged      Nfywrzdeekfn    Scyaprleetwrw   Fabzgpqqn        Cmhkphvvnhv     Eefabsv
Imbcvhs         Okbrexigleft    Momrgzhjuu      Pzzwxnknywv     Jcvrwtp         Fknvsimoi       Qmryoo          Abgxdk           Wawkjelgpkis    Apneohkpcclu
Zztyrkvsfopfkk  Tblle           Csqdeql         Stwbjtf         Kcgtbsusyh      Kjnzstfuzhya    Zzupfjyjvoibj   Ysvxumvkomwaa    Kfdzwddjfhyj    Zltjkxhl
Jdchnlmhwnnw    Tnjyhbiwxwlqy   Qnomeurcclw     Nlnuoz          Ftiml           Nyyvxmjczowcp   Zvcjfulwvzds    Akrlbnneuiat     Dgoigqq         Elhgxqtymbk
Qxatyndiawlft   Leipxrfq        Fefrwvzeba      Wljvlzalvhi     Pqykdehydbao    Zzyznikixvx     Lkpjjrygeexm    Vgqmkhfi         Xpjuv           Uhltzfkmeplrzv

Selection sorting for Word Array (size: 100000) .....
SelectionSort_WordArray(size: 100000) took  18521.013 [millisecond]

After sorting word array (size : 100000)
Aaagj           Aaaminbprqqtl   Aaanjzatlqe     Aaare           Aaaxtltfunz     Aabdcpxzntm     Aabiov          Aabrpbxy         Aabvtshpg       Aaccbghvama
Aacckqsvvpw     Aachwozn        Aackcuy         Aacovtlmfpva    Aacqe           Aadajpejmggbam  Aadbiodkj       Aadcdygzqzf      Aadhrzqkzydm     Aadhyj
Aadpz           Aadwvwo         Aadwzqqtrpm     Aadyborn        Aaedqsdfzlblkx  Aaegwfampm      Aaejticfyycues  Aaenxtpphesg     Aaexex          Aafbkbnwwv
Aafdjzzbsbgqsj  Aafjhiwllzyvsr  Aafmwmxz        Aafnqsvwciy     Aagciuowisreha  Aagifm          Aagjogrvn       Aaguwxedqaj      Aahjpu          Aahkeeysgim
Aahnbijr        Aahvpczbfnp     Aahyytxskmkl    Aahzxipvfoyy    Aaibuut         Aaiegztg        Aaiizr          Aajckyvigjs      Aajeckcsuj      Aajeesrkqagzj
. . . .
Zzpjujvko       Zzpnsgickn      Zzprp           Zzqrdtlvyepfcb  Zzqwikmged      Zzqzra          Zzraprdlf       Zzrbct           Zzrevz          Zzrjzcghdin
Zzrkusea        Zzrmeof         Zzrybgi         Zzrzjviad       Zzsnvwuufh      Zzsqamkykk      Zzsqxaq         Zzstfagjpq       Zzsuzz          Zztafu
Zztyrkvsfopfkk  Zztzxhaqgukh    Zzuaem          Zzugctnfxwlqiw  Zzuilluwtnj     Zzujlgtn        Zzupfjyjvoibj   Zzuwehyo         Zzvagtcimi      Zzvcmbcruj
Zzvkculqzged    Zzvlaxdnqzs     Zzwbuwzmju      Zzwck           Zzwiwzrqcro     Zzwkxpzjmjd     Zzwlatkwmn      Zzwmgzbbkhpinm   Zzxjhh          Zzxki
Zzxkwgoya       Zzxoywsqp       Zzydwxha        Zzyjsmyolopb    Zzysmlfpdsf     Zzyznikixvx     Zzzfzr          Zzzjd            Zzzxewssfxmdvz   Zzzyyebmxtqlkv

계속하려면 아무 키나 누르십시오 . . .
```