2021-1 프로그래밍언어 실습 6

6.1 다중 소스 파일 프로그램 구성

1) BigArray.cpp와 BigArray.h 파일을 준비
2) BigArray.cpp 파일에는 배열의 size가 50,000개 이상인 큰 규모의 배열 데이터를 처리하기 위한 함수들을 구현
3) BigArray.h 파일에는 BigArray.cpp 파일에 포함된 함수들의 함수 원형과 필요한 기호 상수 (symbolic constant), 전처리기 지시자 등을 포함

6.2 RAND_MAX (32767) 보다 더 큰 값의 Big Rand 난수 배열 생성 및 출력

1) RAND_MAX (32767) 보다 더 큰 값의 size와 offset가 주어질 때 난수 (random number) 값의 크기가 (0 ~ size - 1) + offset인 중복되지 않는 난수들을 생성하여 배열에 저장하여 주는 함수 genBigRandArray(int *bigArray, int size, int offset)를 작성하라.
2) genBigRandArray() 함수 호출에서 전달되는 bigArray는 동적으로 할당된 메모리 블록의 주소가 전달되며, 정수 (integer)가 size개 저장될 수 있는 공간이며, 배열로 사용될 수 있다.
3) Big Rand 난수 배열을 출력하기 위하여 void printBigArraySample(int *bigArray, int size, int items_per_line, int num_sample_lines) 함수를 작성하라. 이 함수는 주어진 bigArray[] 배열의 첫 부분에서 한 줄에 items_per_line 개씩 num_sample_line 줄을 출력하고, 배열의 마지막 부분에서 한 줄에 items_per_line 개씩 num_sample_line 줄을 출력하며, 중간 부분은 ". . . . ." 표시를 출력한다.
4) genBigRandArray(), printBigArraySample(),fprintBigArraySample() 함수들은 BigArray.cpp 파일에 구현하고, 그 함수 원형은 BigArray.h에 포함시킬 것.

6.3 hybridQuickSelectionSort(int *bigArray, int size)

1) 주어진 정수 배열을 신속하게 정렬하기 위한 hybridQuickSelectionSort(int *bigArray, int size)를 구현하라. 이 정렬 함수는 정렬 대상 배열 구간의 원소 개수가 작을 때는 선택정렬 방식을 사용하고, 정렬 대상 배열 구간의 원소 개수가 많을 때는 퀵 정렬 방식을 사용하여야 한다. 선택 정렬과 퀵 정렬 방식의 선택은 BigArray.h에서 기호 상수로 설정된 QUICK_SELECTION_THRESHOLD 값에 따라 결정하도록 하라.
2) hybridQuickSelectionSort(int *bigArray, int size) 함수의 정렬 시간을 최소화하기 위하여 partition 기능을 구현하기 위한 함수 호출을 사용하지 않고

재귀 함수로 실행되는 _hybridQuickSelectionSort(int int *bigArray, int size, int left, int right, int level) 함수내에 직접 구현할 것.

3) _hybridQuickSelectionSort(int int *bigArray, int size, int left, int right, int level) 함수와 hybridQuickSelectionSort(int *bigArray, int size) 함수를 BigArray.cpp 파일에 포함시키고, 이 함수들의 함수 원형은 BigArray.h에 포함시킬 것.

6.4 프로그램 모듈 실행 시간 측정

1) 큰 규모의 시스템에 포함된 프로그램 모듈의 성능을 평가하기 위하기 위하여, 해당 모듈의 실행 시간을 측정하고, 분석할 수 있어야 한다. 이 때, 실행 시간을 측정하기 위한 방법으로 Windows 환경에서는 Performance Counter를 사용하여 microsecond 단위로 정밀하게 측정할 수 있다. **QueryPerformanceCounter (LARGE_INTEGER & time)**를 사용하여, performance counter의 값을 microsecond 단위로 읽을 수 있다. 이 기능을 사용하여, 지정된 프로그램 모듈이 실행되기 이전의 performance counter값과 해당 모듈의 실행이 끝난 후의 performance counter값을 각각 기록하여, 그 차이를 계산함으로써, 해당 프로그램 모듈의 실행시간을 측정할 수 있다.

2) Performance Counter를 사용하기 위해서는 <Windows.h> 헤더파일을 포함시켜야 하며, 미리 해당 변수들을 선언하여야 한다: LARGE_INTEGER freq, t1, t2; LONGLONG t_diff; double elapsed_time;

3) Performance Counter에서 사용되는 클락 주파수를 읽기 위해서는 QueryPerformance Frequency(&freq); 함수를 사용한다.

4) Performance Counter값을 읽기 위해서는 QueryPerformanceCounter (LARGE_INTEGER & time)를 사용한다.

6.5 파일 입출력

1) fopen() 함수를 사용하여 "output.txt" 파일을 출력 모드로 생성하라.

2) Big Rand 난수 배열을 지정된 파일로 출력하기 위하여 void fprintBigArraySample(FILE *fout, int *bigArray, int size, int items_per_line, int num_sample_lines) 함수를 작성하라. 이 함수는 printBigArraySample(int *bigArray, int size, int items_per_line, int num_sample_lines)와 동일한 기능을 수행하며, 단지 지정된 출력파일 fout으로 출력한다.

3) 동적 배열의 생성, bigArray 난수 배열의 생성 및 정렬 등에서 처리되는 내용 (예를 들어 정렬하기 전의 샘플 데이터, 정렬한 후의 샘플 데이터 등)을 출력파일에 저장하라.

4) 프로그램 실행이 완료되면 출력파일을 fclose() 함수를 사용하여 닫을 것.

6.6 main() 함수와 실행 결과 예시

1) 메뉴 기반으로 다양한 기능 시험 구성

2) main() 함수

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

#define ESC 0x1B

void Compare_Sorting_Algorithms_SmallIntArray(FILE *fout);
void testBigRandArray(FILE *fout);
void PM_Hybrid_QS_SS_IntArray(FILE *fout);

int main()
{
	FILE *fout;
	int 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: Performance Comparison of Selection Sort and Quick Sort for
		     Small Integer Array\n");
		printf("    2: Test Big Rand Array (Array Size: 1,000,000 ~ 10,000,000)\n");
		printf("    3: Performance Measurements of hybrid_QS_SS for Integer
		    Array\n");
		printf("Input menu (-1 to terminate) : ");
		scanf("%d", &menu);
		//printf("\n");
		if (menu == -1)
			break;
		switch (menu)
		{
		case 1:
			Compare_Sorting_Algorithms_SmallIntArray(fout);
			break;
		case 2:
			testBigRandArray(fout);
			break;
		case 3:
			PM_Hybrid_QS_SS_IntArray(fout);
			break;

		default:
			break;
		}
		fflush(fout);
```

```
        }
        fclose(fout);
        return 0;
}

// 다양한 시험을 위한 함수는 이곳에 구현할 것.
void Compare_Sorting_Algorithms_SmallIntArray(FILE *fout)
{  }
void testBigRandArray(FILE *fout)
{  }
void PM_Hybrid_QS_SS_IntArray(FILE *fout)
{  }
```

3) 화면 출력 결과 1 (일부분)

```
Test Array Algorithms  :
   1: Performance Comparison of Selection Sort and Quick Sort for Small Integer Array
   2: Test Big Rand Array (Array Size: 1,000,000 ~ 10,000,000)
   3: Performance Measurements of hybrid_QS_SS for Integer Array
Input menu (-1 to terminate) : 1
Sorting of an integer array (size :   5) :  Quick_Sort took      0.51 [micro-seconds],   Selection_Sort took      0.26 [micro-seconds]
Sorting of an integer array (size :  10) :  Quick_Sort took      0.51 [micro-seconds],   Selection_Sort took      0.51 [micro-seconds]
Sorting of an integer array (size :  15) :  Quick_Sort took      0.77 [micro-seconds],   Selection_Sort took      0.51 [micro-seconds]
Sorting of an integer array (size :  20) :  Quick_Sort took      1.03 [micro-seconds],   Selection_Sort took      1.03 [micro-seconds]
Sorting of an integer array (size :  25) :  Quick_Sort took      1.28 [micro-seconds],   Selection_Sort took      1.28 [micro-seconds]
Sorting of an integer array (size :  30) :  Quick_Sort took      1.80 [micro-seconds],   Selection_Sort took      1.80 [micro-seconds]
Sorting of an integer array (size :  35) :  Quick_Sort took      2.31 [micro-seconds],   Selection_Sort took      2.05 [micro-seconds]
Sorting of an integer array (size :  40) :  Quick_Sort took      2.31 [micro-seconds],   Selection_Sort took      2.31 [micro-seconds]
Sorting of an integer array (size :  45) :  Quick_Sort took      2.57 [micro-seconds],   Selection_Sort took      3.08 [micro-seconds]
Sorting of an integer array (size :  50) :  Quick_Sort took      3.08 [micro-seconds],   Selection_Sort took      3.59 [micro-seconds]
```

4) 화면 출력 결과 2 (일부분)

```
Test Array Algorithms  :
   1: Performance Comparison of Selection Sort and Quick Sort for Small Integer Array
   2: Test Big Rand Array (Array Size: 1,000,000 ~ 10,000,000)
   3: Performance Measurements of hybrid_QS_SS for Integer Array
Input menu (-1 to terminate) : 2
Testing Big Integer Random Arrays(size = 1,000,000 ~ 10,000,000):
Generating Big Integer array (size =  5000000) . . . .
Generated Big Integer array (size =  5000000):
  238223  1494088  4288425  2805356  1612954  4629547  3230935   475725  4865313  3023987
 3835621  2717456   340576  3784502  4314438  3629139  2488101  1325435  2920262  3702696
  . . . . .
 4264417  1855617  4153428  1384934  3494444   752228   498948   559457  3211122  3991267
 3694473  2376121  1492558  1378590   130679  1871181  3284035  2945537  3491117   951674
Sorted Big Integer array (size =  5000000):
       0        1        2        3        4        5        6        7        8        9
      10       11       12       13       14       15       16       17       18       19
  . . . . .
 4999980  4999981  4999982  4999983  4999984  4999985  4999986  4999987  4999988  4999989
 4999990  4999991  4999992  4999993  4999994  4999995  4999996  4999997  4999998  4999999
Generating Big Integer array (size = 10000000) . . . . .
Generated Big Integer array (size = 10000000):
 4130045  3564161  7628186  1385889  5570108  6852449  4800515  4961029  1149414  8062924
 6827986   883645  8607251  7606563   197880  6054103  4694450   769853   974334  2584721
  . . . . .
 4118620  5297760  9644449  6665526  6807168  1003489  7887033  4955757   890393  5500696
 2032401  4421632  3974408  7304926  8195646   130433  5187720  8673742  4904729   989827
Sorted Big Integer array (size = 10000000):
       0        1        2        3        4        5        6        7        8        9
      10       11       12       13       14       15       16       17       18       19
  . . . . .
 9999980  9999981  9999982  9999983  9999984  9999985  9999986  9999987  9999988  9999989
 9999990  9999991  9999992  9999993  9999994  9999995  9999996  9999997  9999998  9999999
```

5) 화면 출력 결과 3 (일부분)

```
Test Array Algorithms  :
   1: Performance Comparison of Selection Sort and Quick Sort for Small Integer Array
   2: Test Big Rand Array (Array Size: 1,000,000 ~ 10,000,000)
   3: Performance Measurements of hybrid_QS_SS for Integer Array
Input menu (-1 to terminate) : 3
hybrid_QS_SS sorting of an integer array (size :  1000000) :  hybrid_QS_SS sorting took     136.15 [milli-seconds]
hybrid_QS_SS sorting of an integer array (size :  2000000) :  hybrid_QS_SS sorting took     284.78 [milli-seconds]
hybrid_QS_SS sorting of an integer array (size :  3000000) :  hybrid_QS_SS sorting took     436.08 [milli-seconds]
hybrid_QS_SS sorting of an integer array (size :  4000000) :  hybrid_QS_SS sorting took     586.87 [milli-seconds]
hybrid_QS_SS sorting of an integer array (size :  5000000) :  hybrid_QS_SS sorting took     740.57 [milli-seconds]
hybrid_QS_SS sorting of an integer array (size :  6000000) :  hybrid_QS_SS sorting took     904.84 [milli-seconds]
hybrid_QS_SS sorting of an integer array (size :  7000000) :  hybrid_QS_SS sorting took    1060.09 [milli-seconds]
hybrid_QS_SS sorting of an integer array (size :  8000000) :  hybrid_QS_SS sorting took    1227.86 [milli-seconds]
hybrid_QS_SS sorting of an integer array (size :  9000000) :  hybrid_QS_SS sorting took    1383.02 [milli-seconds]
hybrid_QS_SS sorting of an integer array (size : 10000000) :  hybrid_QS_SS sorting took    1544.99 [milli-seconds]

Test Array Algorithms  :
   1: Performance Comparison of Selection Sort and Quick Sort for Small Integer Array
   2: Test Big Rand Array (Array Size: 1,000,000 ~ 10,000,000)
   3: Performance Measurements of hybrid_QS_SS for Integer Array
Input menu (-1 to terminate) :
```

<Oral Test>

Q6.1 동적 메모리 할당의 필요성에 대하여 설명하고, 동적 메모리 할당을 사용하여 동적 배열을 생성하는 방법에 대하여 예를 들어 설명하라.

Q6.2 RAND_MAX (32,767) 보다 큰 값인 배열 크기 size와 offset가 주어지면 (0 ~ size-1) + offset 범위의 값을 가지며 중복되지 않는 난수 (random number)들을 생성하여 지정된 동적 배열에 담아주는 void genBigRandArray (int *bigArray, int size, int offset) 함수의 동작 원리를 설명하라.

Q6.3 Windows 운영체제에서 제공하는 Performance Counter를 사용하여 함수의 실행시간을 millisecond와 microsecond 단위로 정밀하게 측정하는 방법에 대하여 예를 들어 설명하라.

Q6.4 hybridQuickSelectionSort(int *bigRandArray, int size) 함수가 어떻게 selection sorting 과 quick sorting의 장점을 활용하여 다양한 배열의 크기에 대하여 빠르게 정렬할 수 있는지에 대하여 상세하게 설명하라.