4. 배열의 통계 분석, Quick Sorting 알고리즘의 구현 및 실행 시간 측정 (30점, 45분)

1) 정수 배열을 퀵 정렬 정렬 (quick sorting) 방식으로 정렬하는 함수 void quickSort(int *array, int size)를 구현하라. 퀵 정렬 함수에서 사용되는 분할 기능을 partition() 함수로 구현할 것.
2) 정렬 함수의 실행 시간을 측정하기 위하여 Windows에서 제공하는 performance counter를 사용하여 millisecond 단위로 측정하도록 하라. QueryPerformance Frequency(&freq)와 QueryPerformanceCounter (LARGE_INTEGER & time) 함수를 사용할 것.
3) quickSort() 함수는 BigArray_Algorithms.cpp 파일에 구현하고, 그 함수 원형은 BigArray_Algorithms.h에 포함시킬 것.
4) 3 문제에서 구현하였던 genBigRandArray(), printBigArraySample() 함수를 함께 사용할 것.
5) main() 함수에서는 10000 ~ 1000000 범위의 배열 크기 size를 입력받아 동적 배열을 생성하고, genBigRandArray() 함수를 호출하여 중복되지 않은 난수 배열을 생성한 후, printBigArraySample() 함수를 사용하여 첫부분과 끝 부분의 샘플을 출력. quickSort() 함수를 호출하여 오름차순으로 정렬한 후, printBigArraySample() 함수를 사용하여 정렬된 배열 내용을 출력할 것. 아울러, quickSort() 함수 실행에서 걸린 시간을 밀리초 (milli-second) 단위로 측정하여 출력할 것.

```
/* 주석문 */
. . .  // 필요한 전처리기 선언

int main()
{
        int* int_array;
        int array_size;
        LARGE_INTEGER freq, t1, t2;
        LONGLONG t_diff;
        double elapsed_time;

        srand(0);
        QueryPerformanceFrequency(&freq);
        printf("Input array size : ");
        scanf("%d", &array_size);
        int_array = (int*)calloc(array_size, sizeof(int));
        genBigRandArray(int_array, array_size, -array_size / 2);
        printf("\nGenerated big random array ...\n");
        printBigArraySample(int_array, array_size, 10, 3);
        getArrayStatistics(int_array, array_size);

        printf("\nQuickSorting big random array ...\n");
        QueryPerformanceCounter(&t1);
        quickSort(int_array, array_size);
        QueryPerformanceCounter(&t2);
        printf("\n... After sorting ...\n");
        printBigArraySample(int_array, array_size, 10, 3);
        t_diff = t2.QuadPart - t1.QuadPart;
        elapsed_time = (double)t_diff / freq.QuadPart;
        printf(" Quick_Sort took %10.2lf [milli-seconds], ", elapsed_time * 1000.0);
       free(int_array);
        return 0;
}
```

6) 실행 결과 (예시)

```
Input array size : 1000000

Generated big random array ...
 -247097   429221  -327563  -363395   -43788   319204   -78663   472337  -363955  -279624
 -126541   116946   300299  -125152   201078  -348882    43707  -470187   197865  -340925
  244222   329151  -108920  -306073  -167030  -483524  -128223  -342367  -496152   -98528
  . . . . .
 -462653  -483195   369807  -468912   234078   256161   170204   116141  -359538  -353101
  -53108   160902    79900  -413714   122310   251306   429507   104313   321868  -410245
  213155   300993   188040   315001  -390148  -226891  -391442   112329    -2549  -455406
Total (1000000) integer data :
min (-500000) max (499999) sum (-500000.00) average (   -0.50) variance (83333333333.79) standard deviation (288675.13)

QuickSorting big random array ...

... After sorting ...
 -500000  -499999  -499998  -499997  -499996  -499995  -499994  -499993  -499992  -499991
 -499990  -499989  -499988  -499987  -499986  -499985  -499984  -499983  -499982  -499981
 -499980  -499979  -499978  -499977  -499976  -499975  -499974  -499973  -499972  -499971
  . . . . .
  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
 Quick_Sort took     125.12 [milli-seconds],
```