Laboratory 08A

CS-102

Spring 2022

A Word About The Random # Generator

- In order to fully test the comparison between different sorting algorithms, we need to sort numbers that are larger than 32,767 (or two to the 15th power minus 1), which is the limitation of the rand() function that we have been using.
- This limitation is overcome by generating two random numbers, **rand1** and **rand2**, each having 15 bits (that sums up to 32,767 if all the bits are 1).
- If we multiply rand1 by 32,768 (two to the 15th power), this is the equivalent of shifting all the bits of rand1 15 places to the left (see graphic on next page).
- Then if we add to that rand2 (which we haven't shifted), we have created a 30 bit random number which now has a maximum possible value of 1,073,741,823 (two to the thirtieth power minus one).
- The net result is that we will be able to sort numbers up to a billion+ using this technique. Since we only plan to go up to 100 million, this is big enough.

rand1 shifted 15 bits to left plus rand2 gives us our 30 bit pseudo-random number.

A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	AA	AB	AC	AD	AE	AF
Power	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
2 raised to Power	1E+09	5E+08	3E+08	1E+08	7E+07	3E+07	2E+07	8E+06	4E+06	2E+06	1E+06	524288	262144	131072	65536	32768	16384	8192	4096	2048	1024	512	256	128	64	32	16	8	4	2	1
Total Value:																	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
32767																	16384	8192	4096	2048	1024	512	256	128	64	32	16	8	4	2	1
Total Value:		1	1	1	1	1	1	1	1	1	1	1	1	1	1	1															
1073709056		5E+08	3E+08	1E+08	7E+07	3E+07	2E+07	8E+06	4E+06	2E+06	1E+06	524288	262144	131072	65536	32768															
Total Value:		1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
1073741823		5E+08	3E+08	1E+08	7E+07	3E+07	2E+07	8E+06	4E+06	2E+06	1E+06	524288	262144	131072	65536	32768	16384	8192	4096	2048	1024	512	256	128	64	32	16	8	4	2	1

- In C++ the long or int type has 32 bits, (bits 0 through 31), of which bit 31 is a sign bit. Bit 30, we don't need, and so don't use.
- This leaves us with rand2 occupying bits represented by columns R through AF, and rand1, which has been shifted occupying C thru Q.

Sorting Algorithms Part I

- Here are three sorting routines that will first create an array of Random Numbers, and then will sort them. We will use Arrays for Part 1.
- You will be using the Bubble Sort, the Selection Sort, and finally, the QuickSort algorithms.
- If you are doing this Lab synchronously, Teams of two will be used:
  - One member of the team will do the Bubble Sort solution;
  - One will do the Selection Sort solution;
  - Then both of you will do the QuickSort solution.
- If you are doing this Lab asynchronously, do the Bubble Sort and QuickSort Solutions.
- When each of you have them running, set the number of elements to:
  - 1,000; 3,000; 10,000; 30,000; 100,000; and 300,000 and record the time that it takes for it to execute. **Note: if you get zero for an elapsed time, do it 10 times and take the average as the effective elapsed time.**
  - Graph each of the three sets of times for each algorithm using the jpeg file provided to you by the Laboratory08A Assignment. You can use a graphics program, like Windows Paint, or Mac Paintbrush.
  - Use red for Bubble Sort, Blue for Selection Sort and green for QuickSort.
  - I would also suggest that you use a small circle for the Bubblesort, a square for the Selection sort, and a small star for the QuickSort.
  - It is not necessary to draw lines connecting all the data points. The progression of data points, as you graph them, will give you a close enough perspective on how the algorithm, you are graphing, performs.
  - Be sure to record the actual time that the sort took to execute on your graph for each datapoint.

Sorting Algorithms Part I

- Regardless of whether you are doing this Lab synchronously, or asynchronously, you will be creating this graph using the attached log-log image.
- If you are doing this Lab synchronously, show your results for part 1 to the Instructor and submit your graph to Canvas. Be sure to include the following:
  - What kind of computer you are using?
  - What IDE you are using?
  - Indicate the data you are graphing (BubbleSort or Selection Sort, & QuickSort).
- If you are doing this Lab asynchronously, simply submit your results to Canvas directly. Be sure to include the following:
  - What kind of computer you are using?
  - What IDE you are using?
  - Indicate the data you are graphing (BubbleSort, and QuickSort).

Sorting Algorithms Part II

- Repeat the exercise, except, rewrite the programs to use Vectors instead of arrays.
- You will be using the Bubble Sort, the Selection Sort, and finally, the QuickSort algorithms.
- If you are doing this Lab synchronously, Teams of two will be used:
  - One member of the team will do the Bubble Sort solution;
  - One will do the Selection Sort solution;
  - Then both of you will do the QuickSort solution.
- If you are doing this Lab asynchronously, do the Bubble Sort and QuickSort Solutions.
- When each of you have them running, set the number of elements to:
  - 1,000; 3,000; 10,000; 30,000; 100,000; and 300,000 and record the time that it takes for it to execute.
  - Graph each of the three sets of times for each algorithm using the jpeg file provided to you by the Laboratory08A Assignment. You can use a graphics program, like Paint in Windows, or Paintbrush for the Mac.
  - Use red for Bubble Sort, Blue for Selection Sort and green for QuickSort.
  - I would also suggest that you use a small circle for the Bubblesort, a square for the Selection sort, and a small star for the QuickSort.

Sorting Algorithms Part II

- Regardless of whether you are doing this Lab synchronously, or asynchronously, you will be creating this graph using the attached log-log image.
- If you are doing this Lab synchronously, show your results for part 2 to the Instructor and submit your graph to Canvas. Be sure to indicate the following:
  - What kind of computer you are using?
  - What IDE you are using?
  - What data you are graphing (BubbleSort or Selection Sort, and QuickSort)?
- If you are doing this Lab asynchronously, simply submit your results for part 2 to Canvas directly. Be sure to include the following:
  - What kind of computer you are using?
  - What IDE you are using?
  - Indicate the data you are graphing (BubbleSort, and QuickSort).

Sorting Algorithms Part III

- Now, using Vectors instead of arrays, using the QuickSort algorithm, get times for the following number of sorts:
    - 1,000,000
    - 3,000,000
    - 10,000,000
    - 30,000,000
    - 100,000,000
    - **Don't try this with a BubbleSort or even a SelectionSort, as they will be too slow.**
    - **Also, don't try this using Arrays either, as our laptops do not have enough contiguous memory to create arrays of this size.**
- How long did it take a QuickSort to sort 100 million pseudo-random numbers?
- Had you attempted to use a BubbleSort to sort 100 million numbers, can you figure out, approximately, how long this would have taken?
- Why would Arrays not work for a Sort of this size on your personal computer?
- What are the advantages and disadvantages of Vectors for large sorts?

Sorting Algorithms Part III

- Regardless of whether you are doing this Lab synchronously, or asynchronously, you will be creating this graph using the attached log-log image.
- If you are doing this Lab synchronously, show your results for part 3 to the Instructor and submit your two graphs to Canvas. Be sure to include the following:
  - What kind of computer you are using?
  - What IDE you are using?
- If you are doing this Lab asynchronously, simply submit your results for part 3 to Canvas directly. Be sure to include the following:
  - What kind of computer you are using?
  - What IDE you are using?

Appendices

The programs you will be using,
And the graph templates you will be employing.

```
// This program will sort MAX_NUM random numbers, using a Bubble Sort, into ascending order, & show the time of execution.
#include <iostream>
#include <cstdlib>
#include <ctime>
#include <random>
#include <chrono>
using namespace std;
void sortArray(int [], int);
int bigRand(int, int);
const int MAX_NUM = 10000;                    // This you will change
const int MAX_VALUE = 100000000;
const int MIN_VALUE = 1;
int main()
{
   int a[MAX_NUM],i;
   int elapsed_msecs;
   unsigned seed = time(0);
   srand(seed);
   for(i=0;i<MAX_NUM;i++)
         a[i]=bigRand(MAX_VALUE, MIN_VALUE);
   cout << "The clock has now started:" << endl << endl;
   auto started = chrono::high_resolution_clock::now();      // Starts the clock
   sortArray(a,MAX_NUM);
   auto done = chrono::high_resolution_clock::now();         // Stops the clock
   elapsed_msecs = chrono::duration_cast<chrono::milliseconds>(done-started).count();
   cout << "Elapsed time = " << elapsed_msecs << " milliseconds." << endl;
   cin.get();
   return 0;
}
```

Bubble Sort Algorithm I

```
void sortArray(int array[], int size)
{
  bool swap;
  int temp;
  do
  {
    swap = false;
    for (int count = 0; count < (size - 1); count++)
    {
      if (array[count] > array[count + 1])
      {
        temp = array[count];
        array[count] = array[count + 1];
        array[count + 1] = temp;
        swap = true;
      }
    }
  } while (swap);
}
int bigRand(int upper, int lower)
{
   const int twoToFifteenth = 32768;
   int rand_num, rand_num1, rand_num2, randnum;
   rand_num1 = 1 + rand() ;
   rand_num2 = rand() ;
   rand_num = twoToFifteenth*rand_num1 + rand_num2;
   randnum = rand_num%(upper - lower) + lower;
   return randnum;
}
```

Bubble Sort Algorithm II

```
// This program will sort MAX_NUM random numbers, using a Selection Sort, into ascending order, & show the time of execution.
#include <iostream>
#include <cstdlib>
#include <ctime>
#include <random>
#include <chrono>
using namespace std;
void selectionSort(int [], int);
int bigRand(int, int);
const int MAX_NUM = 10000;                    // This you will change
const int MAX_VALUE = 100000000;
const int MIN_VALUE = 1;
int main()
{
   int a[MAX_NUM],i;
   int elapsed_msecs;
   unsigned seed = time(0);
   srand(seed);
   for(i=0;i<MAX_NUM;i++)
         a[i]=bigRand(MAX_VALUE, MIN_VALUE);
   cout << "The clock has now started:" << endl << endl;
   auto started = chrono::high_resolution_clock::now();      // Starts the clock
   selectionSort(a,MAX_NUM);
   auto done = chrono::high_resolution_clock::now();         // Stops the clock
   elapsed_msecs = chrono::duration_cast<chrono::milliseconds>(done-started).count();
   cout << "Elapsed time = " << elapsed_msecs << " milliseconds." << endl;
   cin.get();
   return 0;
}
```

Selection Sort Algorithm I

```
void selectionSort(int array[], int size)
{
   int startScan, minIndex, minValue;
   for (startScan = 0; startScan < (size - 1); startScan++)
   {
      minIndex = startScan;
      minValue = array[startScan];
      for(int index = startScan + 1; index < size; index++)
      {
         if (array[index] < minValue)
         {
                  minValue = array[index];
                  minIndex = index;
         }
      }
      array[minIndex] = array[startScan];
      array[startScan] = minValue;
   }
}
int bigRand(int upper, int lower)
{
   const int twoToFifteenth = 32768;
   int rand_num, rand_num1, rand_num2, randnum;
   rand_num1 = 1 + rand() ;
   rand_num2 = rand() ;
   rand_num = twoToFifteenth*rand_num1 + rand_num2;
   randnum = rand_num%(upper - lower) + lower;
   return randnum;
}
```

Selection Sort Algorithm II

```
// This program will sort MAX_NUM random numbers, using arrays & QuickSort, into ascending order, & show the time of execution.
#include <iostream>
#include <cstdlib>
#include <ctime>
#include <random>
#include <chrono>
using namespace std;
void quick_sort(int[],int,int);
int partition(int[],int,int);
int bigRand(int, int);
const int MAX_NUM = 300000;                    // This you will change
const int MAX_VALUE = 100000000;
const int MIN_VALUE = 1;
int main()
{
   int a[MAX_NUM],i;
   int elapsed_msecs;
   unsigned seed = time(0);
   srand(seed);
   for(i=0;i<MAX_NUM;i++)
        a[i]=bigRand(MAX_VALUE, MIN_VALUE);
   cout << "The clock has now started:" << endl << endl;
   auto started = chrono::high_resolution_clock::now();          // Starts the clock
   quick_sort(a,0,MAX_NUM-1);
   auto done = chrono::high_resolution_clock::now();             // Stops the clock
   elapsed_msecs = chrono::duration_cast<chrono::milliseconds>(done-started).count();
   cout << "Elapsed time = " << elapsed_msecs << " milliseconds." << endl;
   cin.get();
   return 0;
}
```

QuickSort Algorithm I

```
int bigRand(int upper, int lower)
{
   const int twoToFifteenth = 32768;
   int rand_num, rand_num1, rand_num2, randnum;
   rand_num1 = 1 + rand() ;
   rand_num2 = rand() ;
   rand_num = twoToFifteenth*rand_num1 + rand_num2;
   randnum = rand_num%(upper - lower) + lower;
   return randnum;
}

void quick_sort(int a[],int k,int u)
{
   int j;
   if(k<u)
   {
      j=partition(a,k,u);
      quick_sort(a,k,j-1);
      quick_sort(a,j+1,u);
   }
}
```

QuickSort Algorithm II

```
int partition(int a[],int k,int u)
{
   int v,i,j,temp;
   v=a[k];
   i=k;
   j=u+1;
   do
   {
      do
           i++;
      while(a[i]<v&&i<=u);
      do
           j--;
      while(v<a[j]);
      if(i<j)
      {
           temp=a[i];
           a[i]=a[j];
           a[j]=temp;
      }
   }while(i<j);
   a[k]=a[j];
   a[j]=v;
   return(j);
}
```

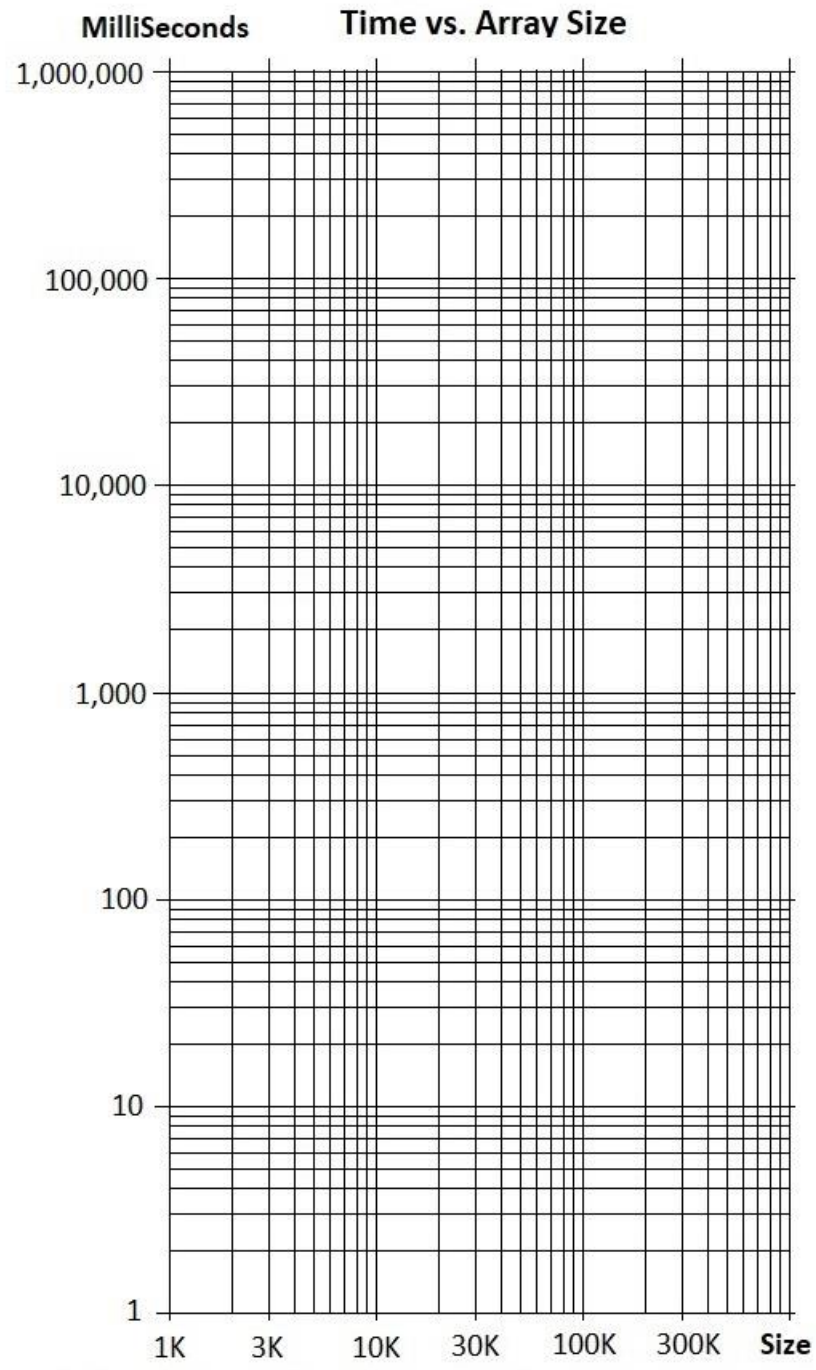
QuickSort Algorithm III

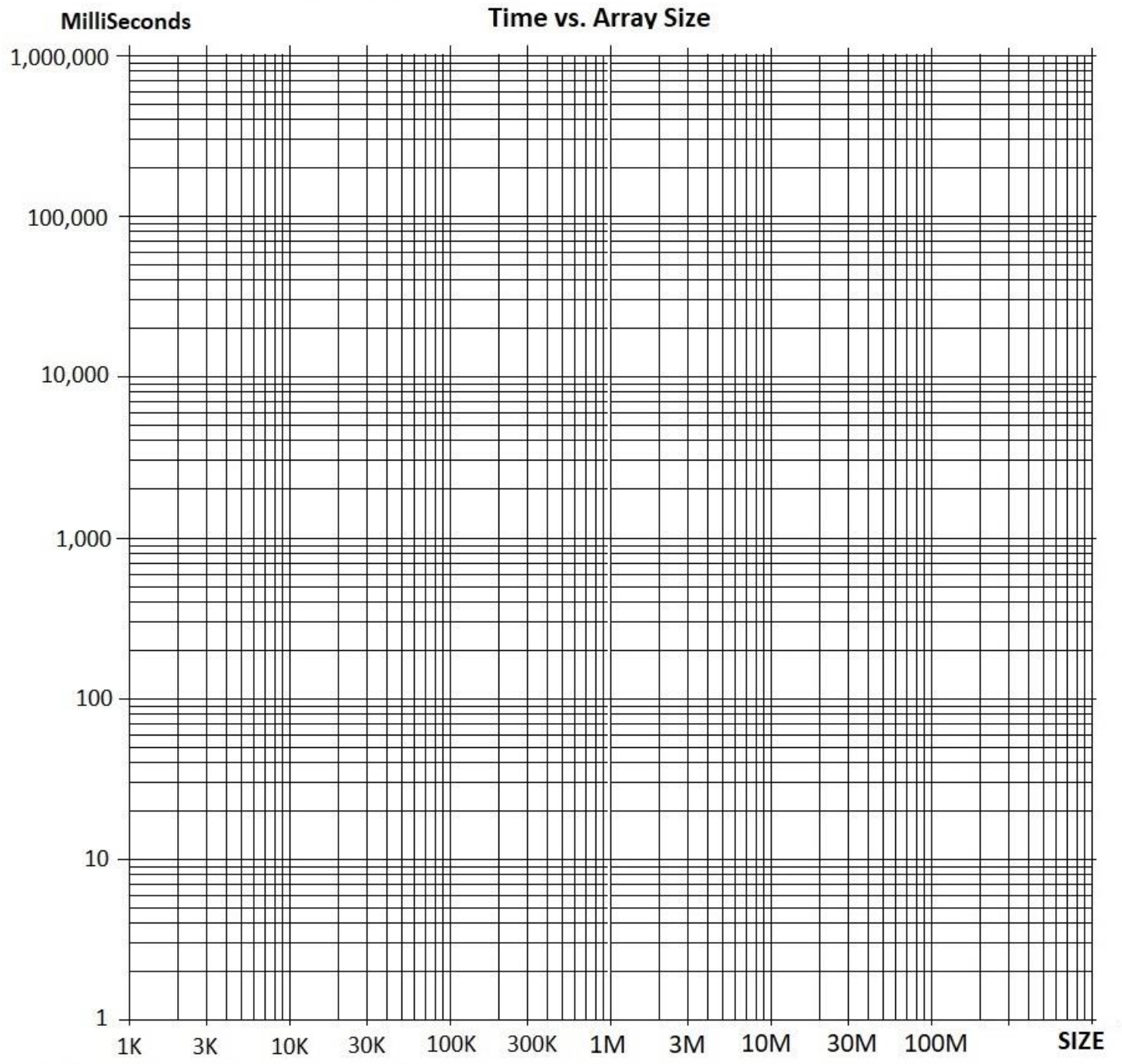
```
// QuickSort - Vector Version
// This program will sort MAX_VALUE random numbers, using a QuickSort
#include <iostream>
#include <cstdlib>
#include <ctime>
#include <vector>
#include <random>
#include <algorithm>
#include <chrono>
using namespace std;
int argc;
char * argv[3];
void commaSeparate(int);
void quick_sort(vector<int>&,int,int);
int partition(vector<int>&,int,int);
int bigRand(int, int);
const int MAX_NUM = 100000000;          // This you will change
const int MAX_VALUE = 100000000;
const int MIN_VALUE = 1;
```

```
int main()
{
    vector<int> a(MAX_VALUE);
    int n,i;
    int elapsed_secs;
    unsigned seed = time(0);
    srand(seed);
    n = MAX_NUM;
    for(i=0;i<n;i++)
        a[i]= bigRand(MAX_VALUE, MIN_VALUE);
    auto started = chrono::high_resolution_clock::now();  // Starts the clock
    quick_sort(a,0,n-1);
    auto done = chrono::high_resolution_clock::now();     // Stops the clock
    elapsed_secs = chrono::duration_cast<chrono::milliseconds>(done-started).count();
    cout << "Elapsed time = " << elapsed_secs << " milliseconds." << endl;
    return 0;
}
```

```
int bigRand(int upper, int lower)
{
    const int twoToFifteenth = 32768;
    int rand_num, rand_num1, rand_num2, randnum;
    rand_num1 = 1 + rand();
    rand_num2 = rand();
    rand_num = twoToFifteenth*rand_num1 + rand_num2;
    randnum = rand_num%(upper - lower) + lower;
    return randnum;
}
void quick_sort(vector<int> &a,int l,int u)
{
    int j;
    if(l<u)
    {
        j=partition(a,l,u);
        quick_sort(a,l,j-1);
        quick_sort(a,j+1,u);
    }
}
```

```
int partition(vector<int> &a,int l,int u)
{
    int v,i,j,temp;
    v=a[l];
    i=l;
    j=u+1;
    do
    {
        do
            i++;
        while(a[i]<v&&i<=u);
        do
            j--;
        while(v<a[j]);
        if(i<j)
        {
            temp=a[i];
            a[i]=a[j];
            a[j]=temp;
        }
    }while(i<j);
    a[l]=a[j];
    a[j]=v;
    return(j);
}
```


MilliSeconds
Time vs. Array Size
1,000,000
100,000
10,000
1,000
100
10
1
1K
3K
10K
30K
100K
300K
Size


Time vs. Array Size
MilliSeconds
1,000,000
100,000
10,000
1,000
100
10
1
1K
3K
10K
30K
100K
300K
1M
3M
10M
30M
100M
SIZE