Code for problems 3-5 are attached at the end

1. Find C_1 and C_2 and n_0 to show that $5x^2 + 4x - 8320$ is $\Theta(n^2)$
   Solution:

$$c1 * x^2 \leq 5x^2 + 4x - 8320 <= c2 * x^2$$

For $n_0 = 1$, c_1 ~= -8311, but it should be positive

X intercept = 40.39, must be greater than that!
let's round up to $n_0 = 50$ →F(50) = 4380, but when we divide by x^2

$$c1 \leq 5 + \frac{4}{x} - \frac{8320}{x^2} \leq c2$$

Now:

Need a bigger $n_0 = 8230$

$c1 \leq \frac{4}{x} - \frac{8320}{x^2}$ for $n>n_0$

$$c1 \leq \frac{1}{x} - (4 - \frac{8320}{x})$$

$$c1 \leq \frac{1}{8320} * (4 - \frac{8320}{8320})$$

$$c1 \leq \frac{3}{8320}$$

C_1 <= 0.00036057692

Now:

$$5 + \frac{4}{x} \leq c2$$

$$5 + \frac{4}{8320} \leq c2$$

5.00048076923 <= C_2

One Solution:

N_0 = 8230

C_1 <= 0.00036057692

C_2 >= 5.00048076923

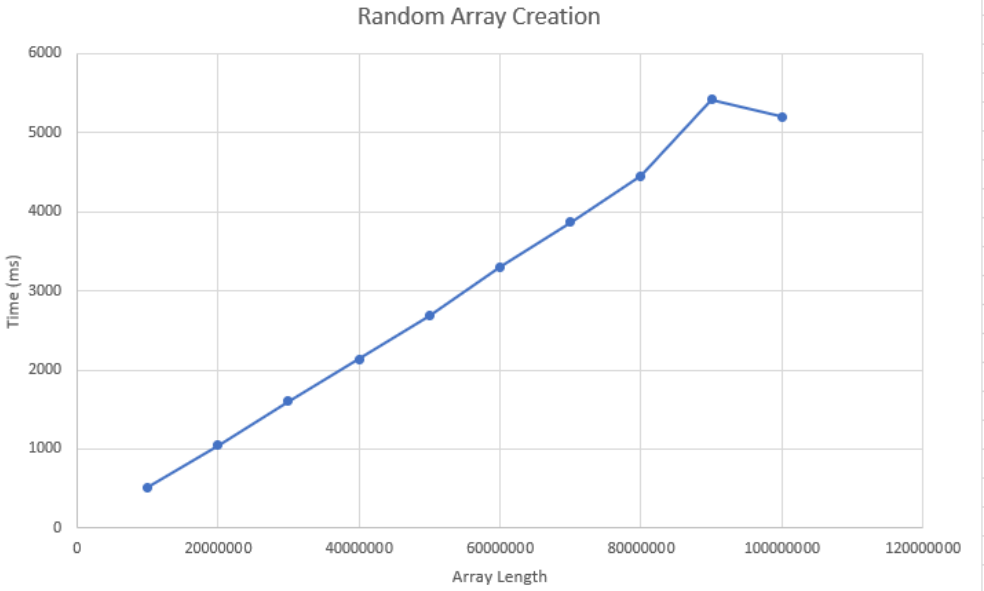
Code for problems 3-5 are attached at the end

2. An algorithm can process 512 items in 2 seconds
   6144/514 = 12 → 12t
   a. Time for 6144 items if algorithm is is $\Theta(n^2)$
      $2*12^2 = 288$ s
   b. Time for 6144 items if algorithm is is $\Theta(n)$
      $2*12 = 24$ s
   c. Time for 6144 items if algorithm is is $\Theta(n^3)$
      $2*12^3 = 3456$ s
   d. Time for 6144 items if algorithm is is $\Theta(2^n)$
      $2*2^{12} = 2^{13}$ s
   e. Time for 6144 items if algorithm is is $\Theta(n^{1/2})$
      2*sqrt(12) = 6.928 s

Code for problems 3-5 are attached at the end

3. Write a program – See Attached Program

	No Sorting
Arr. Len.	Time (ms)
10000000	518
20000000	1044
30000000	1605
40000000	2137
50000000	2682
60000000	3305
70000000	3865
80000000	4455
90000000	5424
100000000	5198



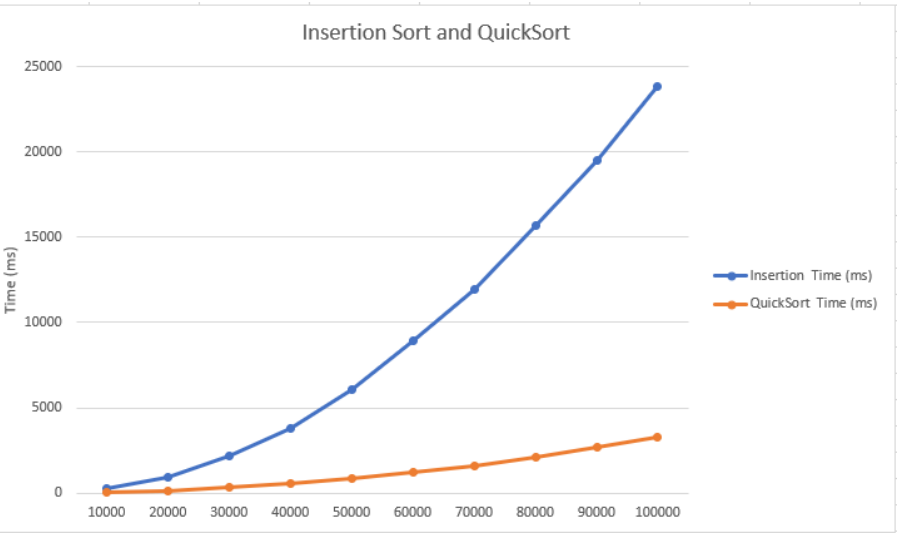
This is an O(n) algorithm whose completion time y (milliseconds) can be predicted by:
y = 0.00005626485*x - 71.26667, where x is the length of the array
To solve for items generated in 3 days, we must solve for X where Y = 3days
1000ms/s*60s/m*60m/hr*24hr/d*3d = 259200000ms

259200000 = 0.00005626485*x - 71.26667
X = 4,606,785,199,759 numbers

Code for problems 3-5 are attached at the end

4. Insertion Sort - See Attached Program

Arr. Len.	Insertion Time (ms)	QuickSort Time (ms)
10000	236	34
20000	942	132
30000	2146	304
40000	3819	528
50000	6099	824
60000	8910	1192
70000	11910	1616
80000	15721	2110
90000	19523	2666
100000	23868	3285

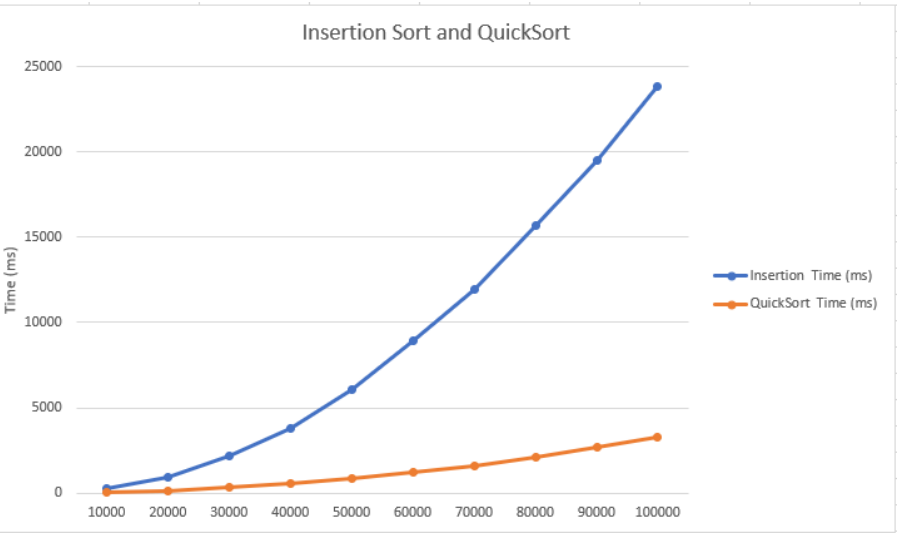


This is an $O(n^2)$ algorithm whose completion time y (milliseconds) can be predicted by:
$y = -234.4 + 0.01334788*x + 0.000002290303*x^2$, where x is the length of the array
To solve for items generated in 3 days, we must solve for X where Y = 3days
3 days = 1000ms/s*60s/m*60m/hr*24hr/d*3d = 259200000ms
$259200000 = -234.4 + 0.01334788*x + 0.000002290303*x^2$
X = 10,635,362 numbers

Code for problems 3-5 are attached at the end

5. QuickSort - See Attached Program

Arr. Len.	Insertion Time (ms)	QuickSort Time (ms)
10000	236	34
20000	942	132
30000	2146	304
40000	3819	528
50000	6099	824
60000	8910	1192
70000	11910	1616
80000	15721	2110
90000	19523	2666
100000	23868	3285



This is a O(n*log(n)) algorithm. A 3 day approximation is harder to predict via a regression extrapolation.

We need to model a function f(n) = C*n*log(n)
Solving for c for y=23868, x=100000… C =
23868 = C * 100000* $\log_2(100000)$
C = 0.01436996787

We can get a rough estimate for the array length by solving for x:
259200000ms = 0.01436996787*x*log(x)
X = 876,000,000 numbers

Code for problems 3-5 are attached at the end

6. Given the table, fill in the bounds you can

Highlighted →Given in Assignment

←	Problem	→	←	Algorithm	→	←	Solution	→
Best	Average	Worst	Best	Average	Worst	Best	Average	Worst
				$\Omega(n)$	$O(n)$		$\Omega(n)$	$O(n)$
$\Omega(n)$			$\Theta(n^2)$			$\Omega(n^2)$	$O(n^2)$	
$\Omega(n^2)$	$\Theta(n^2)$		$\Omega(n^2)$	$\Theta(n^2)$			$\Omega(n^2)$	

```
 1  using System;
 2  using System.Diagnostics;
 3  using System.Collections.Generic;
 4  using System.Linq;
 5
 6  namespace Homework1
 7  {
 8      class Program
 9      {
10          enum Sortings { None, Insertion, Quick };
11
12          //Presets
13          const int MIN_RANDOM = 1;
14          const int MAX_RANDOM = 10;
15          const Sortings SORT_FUNCTION = Sortings.Quick;
16
17
18          static void Main(string[] args)
19          {
20              //Test();
21              //Environment.Exit(0);
22              var input = GetSizeInput(args.ElementAtOrDefault(0));
23
24              if (input == -1)
25              {
26                  RunSequence();
27              }
28              else
29              {
30                  PrintHeaders();
31                  RunIteration(input, Sortings.None);
32              }
33          }
34
35          static void PrintHeaders()
36          {
37              Console.Write(" Arr. Len., Time (ms),");
38              for (int i = MIN_RANDOM; i < MAX_RANDOM + 1; i++)
39              {
40                  Console.Write("{0,10},", i);
41              }
42              Console.WriteLine();
43          }
44
45          static void RunSequence()
46          {
47              Console.WriteLine("No Sorting, 10-100M numbers:");
48              PrintHeaders();
49              for (int i = 1; i <= 10; i++)
50              {
51                  RunIteration(i * 10000000, Sortings.None);
52              }
```

```
53
54              Console.WriteLine("Insertion Sorting, 10k-100k numbers:");
55              PrintHeaders();
56              for (int i = 1; i <= 10; i++)
57              {
58                  RunIteration(i * 10000, Sortings.Insertion);
59              }
60
61              Console.WriteLine("Quick Sorting, 10k-100k numbers:");
62              PrintHeaders();
63              for (int i = 1; i <= 10; i++)
64              {
65                  RunIteration(i * 10000, Sortings.Quick);
66              }
67          }
68
69          static void RunIteration(int length, Sortings function)
70          {
71              Stopwatch stopwatch = new Stopwatch();
72              stopwatch.Start();
73              var randoms = GenerateRandomArray(length, function);
74              var hist = GenerateHistogram(randoms);
75              stopwatch.Stop();
76
77              Console.Write("{0,10},{1,10},", length,                    ↵
                  stopwatch.ElapsedMilliseconds);
78              for (int i = MIN_RANDOM; i < MAX_RANDOM + 1; i++)
79              {
80                  Console.Write("{0,10},", hist.GetValueOrDefault(i, 0));
81              }
82
83              Console.WriteLine();
84          }
85
86          static int GetSizeInput(string raw)
87          {
88              if (int.TryParse(raw, out int parsed))
89              {
90                  return parsed;
91              }
92              else
93              {
94                  return -1;
95              }
96          }
97
98          static int[] GenerateRandomArray(int length, Sortings function)
99          {
100             int[] randoms = new int[length];
101             Random factory = new Random();
102             for (int i = 0; i < randoms.Length; i++)
103             {
```

```
104                 randoms[i] = factory.Next(MIN_RANDOM, MAX_RANDOM + 1);
105             }
106 
107 
108             switch(function)
109             {
110                 case Sortings.Insertion:
111                     {
112                         InsertionSort(randoms);
113                         break;
114                     }
115                 case Sortings.Quick:
116                     {
117                         QuickSort(randoms);
118                         break;
119                     }
120             }
121             return randoms;
122         }
123 
124         static Dictionary<int, int> GenerateHistogram(int[] randoms)
125         {
126             var histogram = new Dictionary<int, int>();
127             foreach (int number in randoms)
128             {
129                 if (histogram.ContainsKey(number))
130                 {
131                     histogram[number]++;
132                 }
133                 else
134                 {
135                     histogram.Add(number, 1);
136                 }
137             }
138             return histogram;
139         }
140 
141         static void Test()
142         {
143             Console.WriteLine("Running Test:");
144             int[] randoms = GenerateRandomArray(20, SORT_FUNCTION);
145             var hist = GenerateHistogram(randoms);
146 
147             InsertionSort(randoms);
148             foreach (int i in randoms)
149             {
150                 Console.WriteLine(i);
151             }
152         }
153 
154         //Sorting Algorithms
155         static int[] InsertionSort(int[] array)
```

```
156         {
157             for (int outer = 0; outer < array.Length - 1; outer++)
158             {
159                 for (int inner = outer + 1; inner > 0; inner--)
160                 {
161                     if (array[inner - 1] > array[inner])
162                     {
163                         int swap = array[inner - 1];
164                         array[inner - 1] = array[inner];
165                         array[inner] = swap;
166                     }
167                 }
168             }
169             return array;
170         }
171 
172         // QuickSort/3 and Partition/3 were taken from:
173         // http://csharpexamples.com/c-quick-sort-algorithm-implementation/
174         // I wrote QuickSort/1
175 
176         static void QuickSort(int[] arr)
177         {
178             QuickSort(arr, 0, arr.Length - 1);
179         }
180 
181         static void QuickSort(int[] arr, int start, int end)
182         {
183             int i;
184             if (start < end)
185             {
186                 i = Partition(arr, start, end);
187 
188                 QuickSort(arr, start, i - 1);
189                 QuickSort(arr, i + 1, end);
190             }
191         }
192 
193         static int Partition(int[] arr, int start, int end)
194         {
195             int temp;
196             int p = arr[end];
197             int i = start - 1;
198 
199             for (int j = start; j <= end - 1; j++)
200             {
201                 if (arr[j] <= p)
202                 {
203                     i++;
204                     temp = arr[i];
205                     arr[i] = arr[j];
206                     arr[j] = temp;
207                 }
```

```
208             }
209
210             temp = arr[i + 1];
211             arr[i + 1] = arr[end];
212             arr[end] = temp;
213             return i + 1;
214         }
215     }
216 }
217
```