Steven Imle ski0005@auburn.edu

Files

```
info
 |- pseudocode.txt
 |- specification.pdf
 |- strategies.md

src
 |- strat_1
 |   |- Strat1.cpp
 |   \- Strat1.h
 |- strat_2
 |   |- Strat2.cpp
 |   \- Strat2.h
 |- strat_3
 |   |- Strat3.cpp
 |   \- Strat3.h
 |- AbstractStrategy.h
 |- ArgParser.cpp
 |- ArgParser.h
 |- err_codes.cpp
 |- err_codes.h
 |- FileHandler.cpp
 |- FileHandler.h
 |- Logger.cpp
 |- Logger.h
 |- StrategyTester.cpp
 |- StrategyTester.h
 |- ValueGenerator.cpp
 \- ValueGenerator.h
```

Compilation

The source code for this project was developed in CLion 2016.3 EAP.
The project can be compiled by typing `make` in the root directory.

Steven Imle ski0005@auburn.edu

Ranking

1. Strategy 3
2. Strategy 2
3. Strategy 1

These algorithms were developed in the order they are numbered but ended up becoming more efficient per each new strategy.

Strategy 3 is clearly the least efficient as indicated by the graph but also because no matter what it will always test every possible combination of numbers regardless of whether the number it starts or ends on is negative.

Strategy 2's worst case shows that while it is not always more efficient than strategy 3, it will generally be an order of magnitude faster than it with larger sets of numbers as long as there is a relatively unskewed distribution of numbers.

Strategy 1 is far and above the winner. Since it runs in linear time.

Plagiarism

NO-PLAGIARISM CERTIFICATION: I certify that I wrote the code I am submitting. I did not copy whole or parts of it
from another student or have another person write the code for me. Any code I am reusing in my program is clearly
marked as such with its source clearly identified in comments.

Steven Imle ski0005@auburn.edu

Strategy 1 - (Self Designed)

Iterate over the list of integers starting from the first value in the array to the last value. Using two loops, start by pointing them both at the first value and then incrementing the inner loop by one until the end of the array is reached. When a sum is calculated, compare it to the current value of the smallest known sum and if it is smaller, overwrite it and update the indices of the range.

Pseudo-code

```
algorithm-1 (A: Array [1 ... n] of integer)
  1       smallest := A[0]
  2       ind_l := 0
  3       ind_r := 0
  4       current := 0
  5       for i := 1 to (n-1)
  6           for j := i to (n-1)
  7               for k := i to j
  8                   current := current + A[k]
  9               if smallest = nil or smallest > current
  10                  smallest := current
  11                  ind_l := i
  12                  ind_r := j
  13              current := 0
  14      return (smallest, ind_l, ind_r)
```

Steven Imle ski0005@auburn.edu

Time Complexity

	Cost	Runs	Total
1	1	1	1
2	1	1	1
3	1	1	1
4	1	1	1
5	1	n	n
6	1	n(n + 1) / 2 - 1	n(n + 1)/2 - 1
7	1	2n^3 - 3n^2 + n	2n^3 - 3n^2 + n
8	6	2n^3 - 7/2n^2 + 3/2n	6(2n^3 - 7/2n^2 + 3/2n)
9	5	2n^3 - 3n^2 + n	5(2n^3 - 3n^2 + n)
10	2	2n^3 - 3n^2 + n	2(2n^3 - 3n^2 + n)
11	2	2n^3 - 3n^2 + n	2(2n^3 - 3n^2 + n)
12	2	2n^3 - 3n^2 + n	2(2n^3 - 3n^2 + n)
13	1	2n^3 - 3n^2 + n	2n^3 - 3n^2 + n
14	1	1	1

T(n) = 1/2 * (76*n^3 - 119*n^2 + 47*n + 8)

Steven Imle ski0005@auburn.edu

Strategy 2 - (Self Designed)

Pass over the array one time and add the index of any negative integers into an array. Iterate over these indices using the process from strategy 1 using the indices that are in the array since adding a positive number as the last or first number in the series will not yield a lower result.

Pseudo-code

```
algorithm-2 (A: Array [1 ... n] of integer)
  1       negatives := []
  2       m = 0
  3       for i := 1 to (n-1)
  4           if A[i] < 0
  5               negatives.append(i)
  6               m = m + 1
  7           if A[i] < smallest
  8               smallest := A[i]
  9               ind_l := i
  10              ind_r := i
  11      smallest := nil
  12      ind_l := nil
  13      ind_r := nil
  14      current := 0
  15      for i := 1 to (m-1)
  16          for j := i to (m-1)
  17              for k := negatives[i] to negatives[j]
  18                  current := current + A[k]
  19              if smallest = nil or smallest > current
  20                  smallest := current
  21                  ind_l := i
  22                  ind_r := j
  23              current := 0
  24      return (smallest, ind_l, ind_r)
```

Time Complexity

This strategies time complexity is difficult to calculate since the size of `m` is relative to the amount of negative integers in `A`. Since this can be anywhere from 0 to `n`, I calculated the worst case. This will likely be different from the curve actually calculated since the average case (on a random set of integers) will likely have `n/2` negative numbers in it. Due to this, I have included the upper and lower bounds below.

Ω(n) = n
O(n) = n^3

Steven Imle ski0005@auburn.edu

	Cost	Runs	Total
1	1	1	1
2	1	1	1
3	1	n	n
4	4	n - 1	4n - 4
5	1	n - 1	n - 1
6	3	n - 1	3n - 3
7	5	n - 1	5n - 5
8	4	n - 1	4n - 4
9	2	n - 1	2n - 2
10	2	n - 1	2n - 2
11	1	1	1
12	1	1	1
13	1	1	1
14	1	1	1
15	1	n	n
16	1	n(n + 1)/2 - 1	n(n + 1)/2 - 1
17	1	2n^3 - 3n^2 + n	2n^3 - 3n^2 + n
18	6	2n^3 - 7/2n^2 + 3/2n	6 (2n^3 - 7/2n^2 + 3/2n)
19	5	2n^3 - 3n^2 + n	5 (2n^3 - 3n^2 + n)
20	2	2n^3 - 3n^2 + n	2 (2n^3 - 3n^2 + n)
21	2	2n^3 - 3n^2 + n	2 (2n^3 - 3n^2 + n)
22	1	2n^3 - 3n^2 + n	2n^3 - 3n^2 + n
23	1	2n^3 - 3n^2 + n	2n^3 - 3n^2 + n
24	1	1	1

WORST CASE
T(n) = 1/2 * (72*n^3 - 113*n^2 + 89*n - 28)

LIKELY AVERAGE CASE
T(n) = 1/2 * (72*(n/2)^3 - 113*(n/2)^2 + 89*(n/2) - 28)

Steven Imle ski0005@auburn.edu

Strategy 3 - (Reverse of Kadane's Algorithm)

Walk through the array and find the first negative value, set the starting point to this index and the end point to the same. Add the next value to it. If the sum of the two numbers is less than 0, then increment the end point and change the current smallest sum to the most recently calculated one. If the sum were greater than or equal to 0, do not change anything but step through the array until a new negative number is found and perform the same steps, only changing the value of smallest if the current sum is smaller.

Pseudo-code

```
algorithm-3 (A: Array [l ... r] of integer)
  1       smallest_ending_here := 0
  2       ind_l_ending_here := nil
  3       smallest := 0
  4       ind_l := nil
  5       ind_r := nil
  6       has_negative := false
  7       for i := 1 to (n-1)
  8           if A[i] < 0
  9               has_negative := true
  10              break
  11          if A[i] < smallest
  12              smallest := A[i]
  13              ind_l := i
  14              ind_r := i
  15      if !has_negative
  16          return (smallest, ind_l, ind_r)
  17      for i := 1 to (n-1)
  18          smallest_ending_here = smallest_ending_here + A[i]
  19          if smallest_ending_here > 0
  20              smallest_ending_here = 0
  21              ind_l_ending_here = i + 1
  22          if smallest > smallest_ending_here
  23              smallest := smallest_ending_here
  24              ind_l := ind_l_ending_here
  25              ind_r := i
  26          if smallest > A[i]
  27              smallest := A[i]
  28      return (smallest, ind_l, ind_r)
```

Steven Imle ski0005@auburn.edu

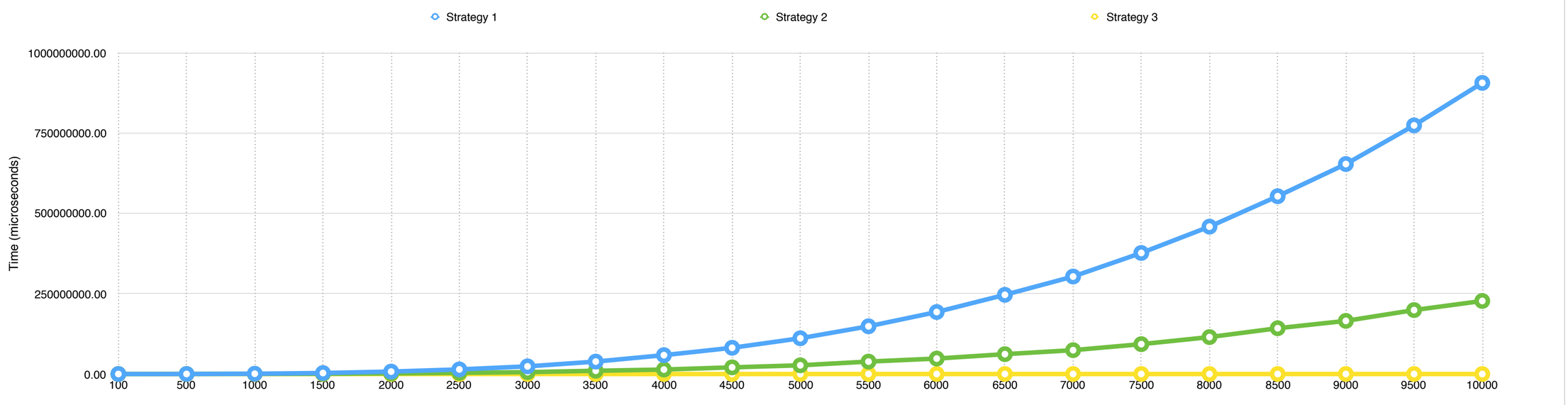
Time Complexity

	Cost	Runs	Total
1	1	1	1
2	1	1	1
3	1	1	1
4	1	1	1
5	1	1	1
6	1	1	1
7	1	n + 1	n + 1
8	4	n	4n
9	1	n	n
10	1	n	n
11	5	n	5n
12	4	n	4n
13	2	n	2n
14	2	n	2n
15	2	1	2
16	1	1	1
17	1	n + 1	n + 1
18	6	n	6n
19	2	n	2n
20	1	n	n
21	3	n	3n
22	3	n	3n
23	2	n	2n
24	2	n	2n
25	2	n	2n
26	5	n	5n
27	4	n	4n
28	1	1	1

$T(n) = 12 + 51n$

Strategy Analysis

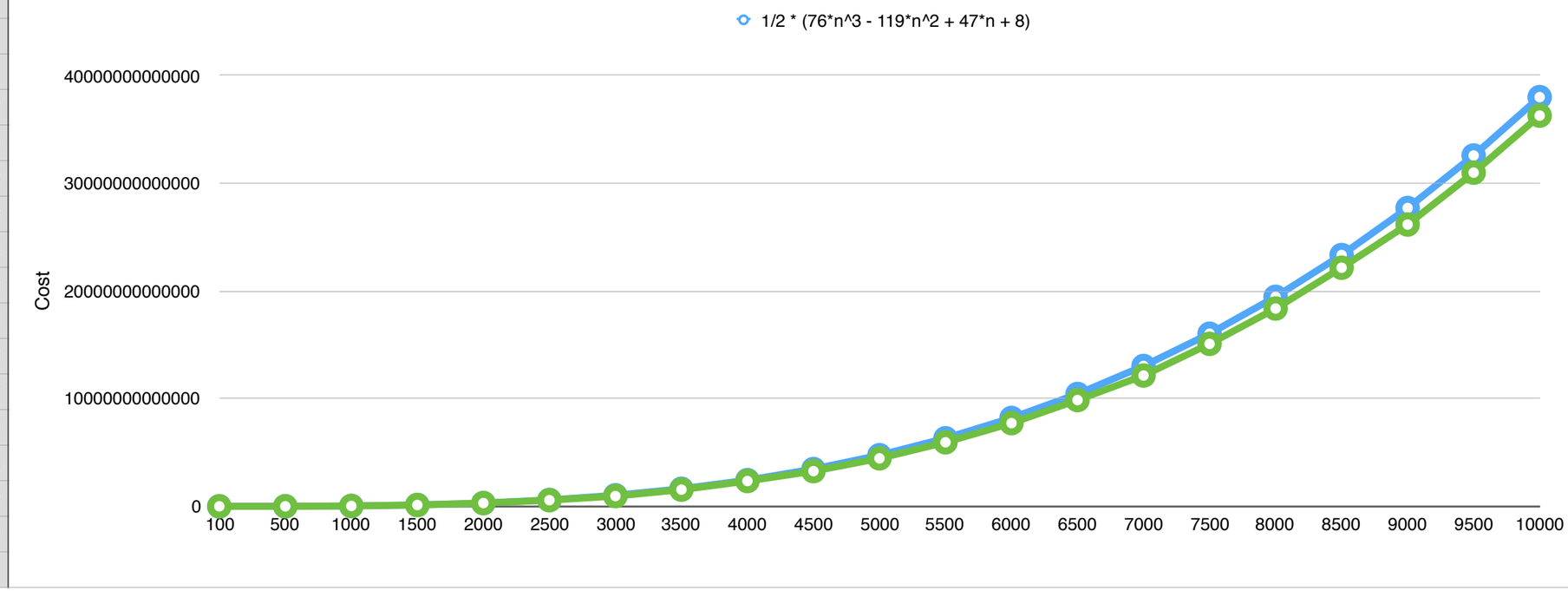
	100	500	1000	1500	2000	2500	3000	3500	4000	4500	5000	5500	6000	6500	7000	7500	8000	8500	9000	9500	10000
Strategy 1	1078.00	106570.00	824681.00	3076137.00	7539485.00	14302282.00	23772165.00	38689812.00	58708986.00	81653325.00	111333450.00	148637635.00	193118154.00	246647266.00	303639434.00	377106028.00	459102276.00	553858484.00	654090578.00	774563653.00	906928851.00
Strategy 2	199.00	28286.00	238359.00	694493.00	1972358.00	3634524.00	5827999.00	10081976.00	13873280.00	20700874.00	27084668.00	38674814.00	48112816.00	61806162.00	74237256.00	92986242.00	114990858.00	142609231.00	165488788.00	199141784.00	227548321.00
Strategy 3	2.03	12.20	19.58	20.13	22.52	26.06	31.72	36.67	41.63	50.32	67.79	71.74	72.62	72.01	81.99	89.10	92.10	97.97	99.59	121.27	115.81



Strategy 1

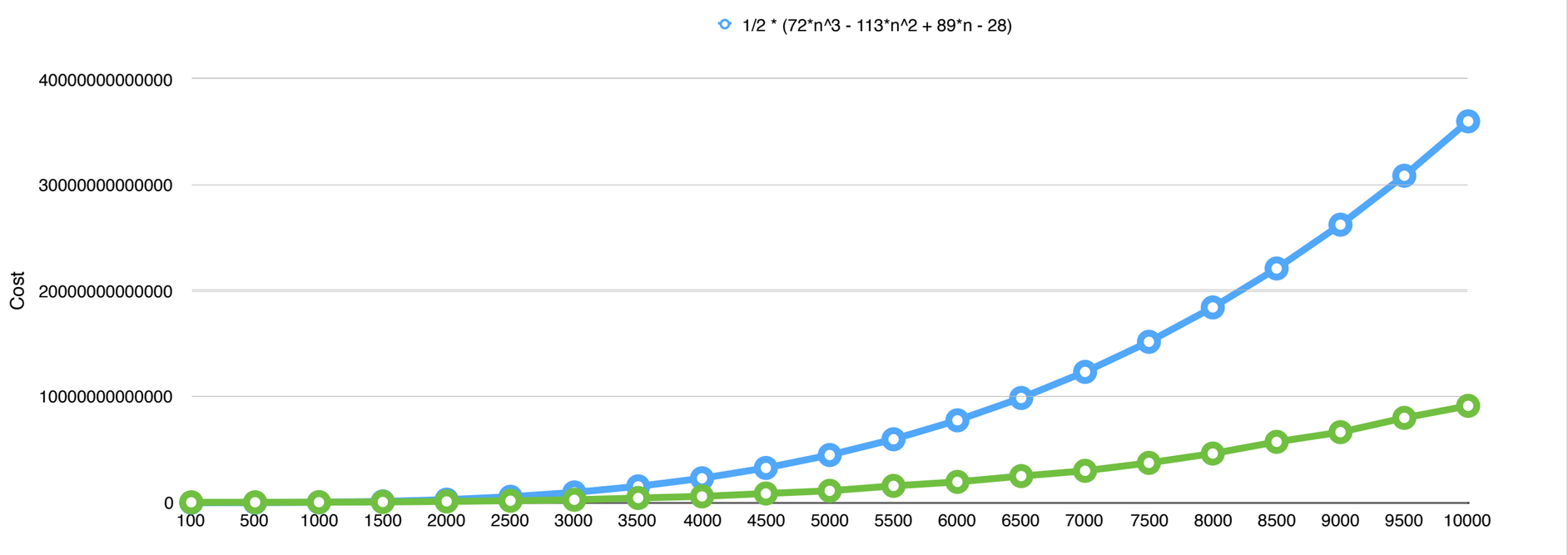
	100	500	1000	1500	2000	2500	3000	3500	4000	4500	5000
1/2 * (76*n^3 - 119*n^2 + 47*n + 8)	37407354	4735136754	37940523504	128116160254	303762047004	593378183754	1025464570504	1628521207254	2431048094004	3461545230754	4748512617504
	5500	6000	6500	7000	7500	8000	8500	9000	9500	10000	
1/2 * (76*n^3 - 119*n^2 + 47*n + 8)	6320450254254	8205858141004	10433236277754	13031084664504	16027903301254	19452192188004	23332451324754	27697180711504	32574880348254	37994050235004	

2.15
3.41437500261492
5.09696146746419
7.25751279974823
9.95578240466746
13.2515236874223
17.2044900532132
21.8744349072405
27.3211116547047
33.6042737008062
40.7836744507454
48.9190673097228
58.0702056829386



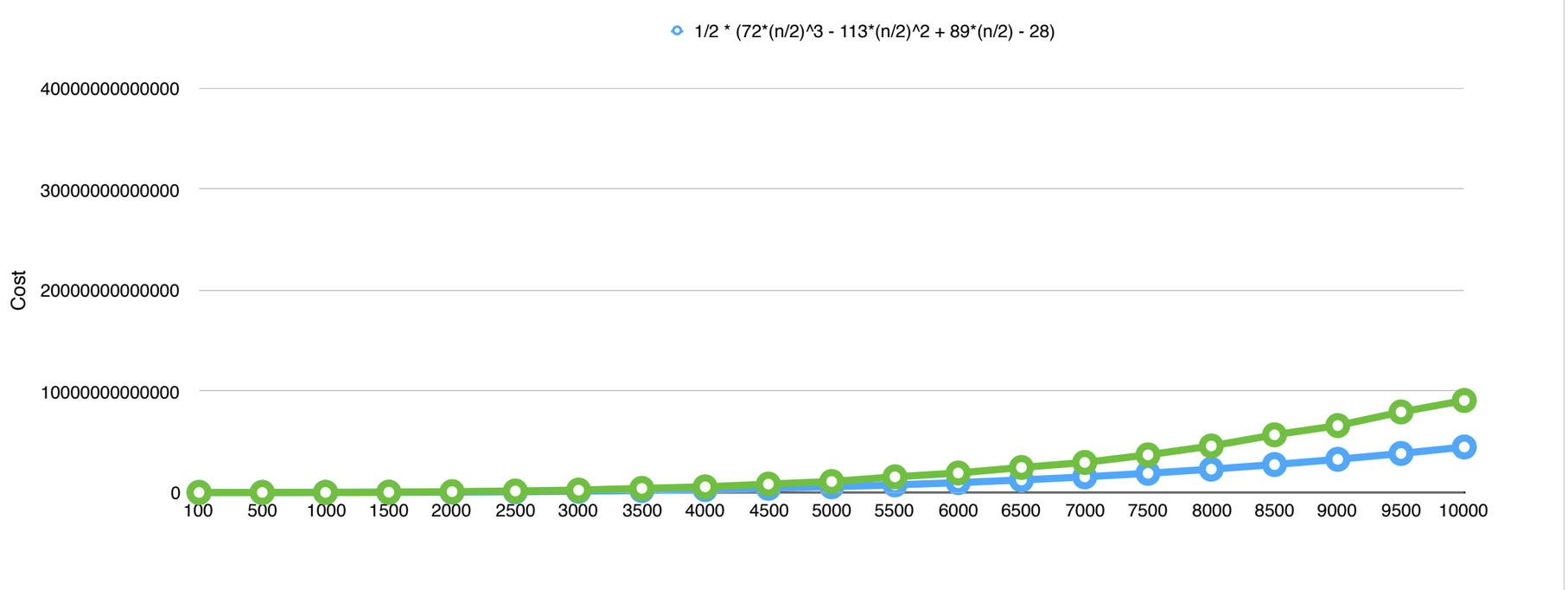
Strategy 2 – Worst Case

	100	500	1000	1500	2000	2500	3000	3500	4000	4500	5000
1/2 * (72*n^3 - 113*n^2 + 89*n - 28)	35439436	4485897236	35943544486	121372941736	287774088986	562146986236	971491633486	1542808030736	2303096177986	3279356075236	4498587722486
	5500	6000	6500	7000	7500	8000	8500	9000	9500	10000	
1/2 * (72*n^3 - 113*n^2 + 89*n - 28)	5987791119736	7773966266986	9884113164236	12345231811486	15184322208736	18428384355986	22104418253236	26239423900486	30860401297736	35994350444986	



Strategy 2 – Likely Average Case

	100	500	1000	1500	2000	2500	3000	3500	4000	4500	5000
1/2 * (72*(n/2)^3 - 113*(n/2)^2 + 89*(n/2) - 28)	4360961	558979861	4485897236	15155752111	35943544486	70224274361	121372941736	192764546611	287774088986	409776568861	562146986236
	5500	6000	6500	7000	7500	8000	8500	9000	9500	10000	
1/2 * (72*(n/2)^3 - 113*(n/2)^2 + 89*(n/2) - 28)	748260341111	971491633486	1235215863361	1542808030736	1897643135611	2303096177986	2762542157861	3279356075236	3856912930111	4498587722486	



Strategy 3

	100	500	1000	1500	2000	2500	3000	3500	4000	4500	5000	5500	6000	6500	7000	7500	8000	8500	9000	9500	10000
12 + 51n	5112	25512	51012	76512	102012	127512	153012	178512	204012	229512	255012	280512	306012	331512	357012	382512	408012	433512	459012	484512	510012