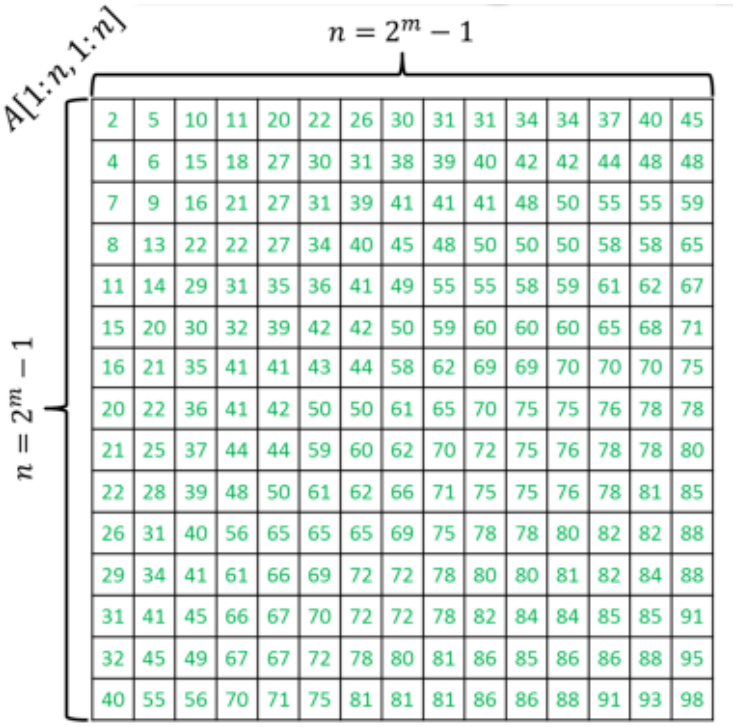
CSE548 & AMS542: Analysis of Algorithms, Fall 2017 **Lecturer:** Rezaul A. Chowdhury
Topic: Asymptotic Bounds **Date:** 08/30/2017
Scribe: Komal Gyanani

1.1 Compare algorithms for asymptotic bounds

Let us analyse and compare four algorithms to solve the same problem with the goal of getting insights on the concept of asymptotic bounds.

Algorithm 1 Searching in a Sorted Grid

- *Scan the grid for the number X, row-by-row or column-by-column, until the grid is exhausted or the number is found and counting only the number of comparisons*
- *Let* $Q_1(n)$ = $number\ of\ comparisons\ in\ the\ n \times n\ grid$
- *In the worst case scenario the number of comparisons are* n^2



2	5	10	11	20	22	26	30	31	31	34	34	37	40	45
4	6	15	18	27	30	31	38	39	40	42	42	44	48	48
7	9	16	21	27	31	39	41	41	41	48	50	55	55	59
8	13	22	22	27	34	40	45	48	50	50	50	58	58	65
11	14	29	31	35	36	41	49	55	55	58	59	61	62	67
15	20	30	32	39	42	42	50	59	60	60	60	65	68	71
16	21	35	41	41	43	44	58	62	69	69	70	70	70	75
20	22	36	41	42	50	50	61	65	70	75	75	76	78	78
21	25	37	44	44	59	60	62	70	72	75	76	78	78	80
22	28	39	48	50	61	62	66	71	75	75	76	78	81	85
26	31	40	56	65	65	65	69	75	78	78	80	82	82	88
29	34	41	61	66	69	72	72	78	80	80	81	82	84	88
31	41	45	66	67	70	72	72	78	82	84	84	85	85	91
32	45	49	67	67	72	78	80	81	86	85	86	86	88	95
40	55	56	70	71	75	81	81	81	86	86	88	91	93	98

Figure 1.1.1: Algorithm 1 Grid Search

Algorithm 2 Searching in a Sorted Grid

- *Since n (dimension of the matrix) is odd, find the central number in the grid and compare X to it*
- *If X <Y, all the numbers in A22, R2 and C2 are greater than the number. Skip A22, R2 and C2.*
- *Search for X in R1 and C1 using binary search.*
- *Recursively search X in A11, A12 and A21.*
- *Let* $Q_2(n)$ = $number\ of\ comparisons\ in\ the\ n \times n\ grid$
- *For simplicity, assume linear scan for R1 and C1, then* $Q_2(n) \leq 1 + 2(\frac{n+1}{2} - 1) + 3Q_2(\frac{n+1}{2} - 1)$
  On Solving: $Q_2(n) \leq 2(n+1)^{log_2 3} \leq 2(n+1)^{1.6}$
  NOTE: *3 comes from the three quadrants and 2 is the reduction factor for each time the grid is processed*

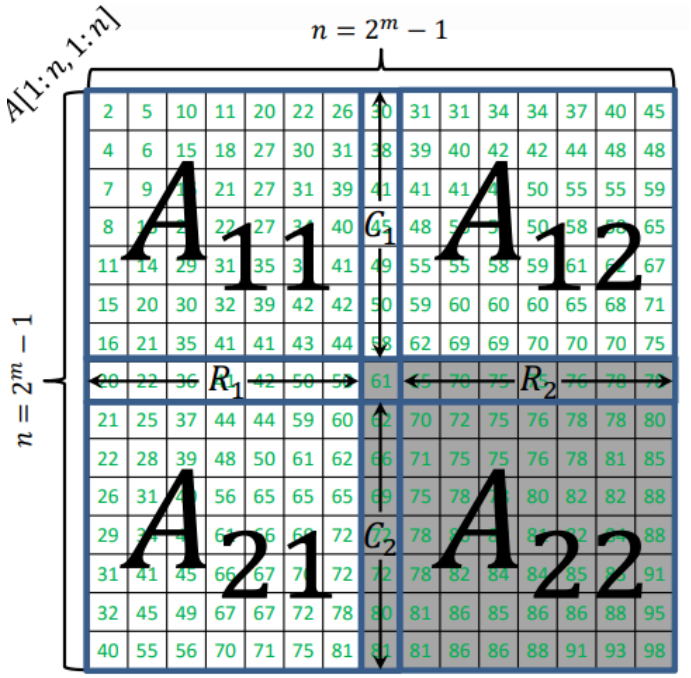
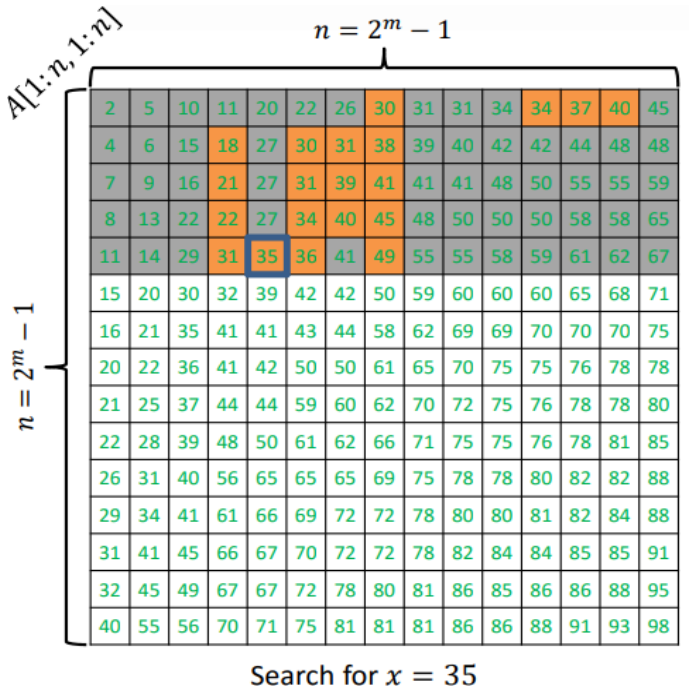


Figure 1.1.2: Algorithm 2 Grid Search

Algorithm 3 Searching in a Sorted Grid

- *Starting from row 1, recursively perform binary search for X in every row of the grid, until the number is found*
- *Let* $Q_3(n)$ = $number\ of\ comparisons\ in\ the\ n \times n\ grid$
- *Then* $Q_3(n) \leq nm = nlog_2(n+1)$

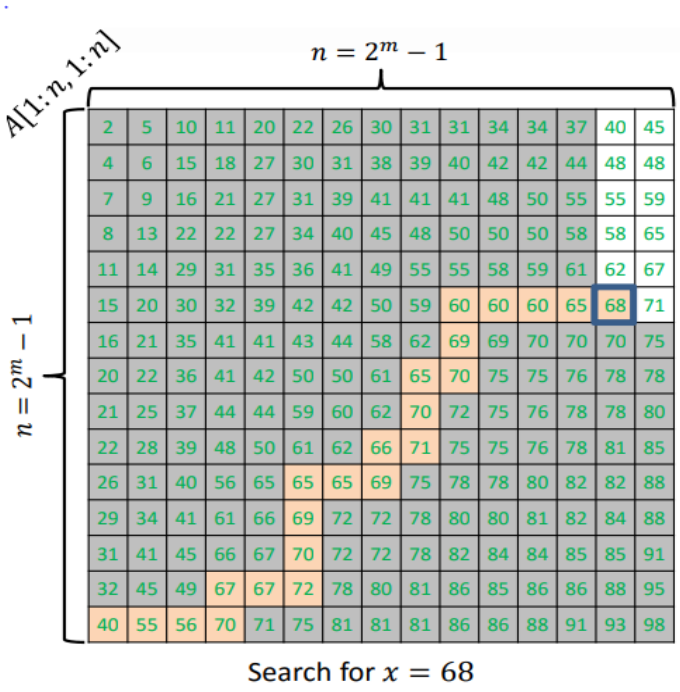


2	5	10	11	20	22	26	30	31	31	34	34	37	40	45
4	6	15	18	27	30	31	38	39	40	42	42	44	48	48
7	9	16	21	27	31	39	41	41	41	48	50	55	55	59
8	13	22	22	27	34	40	45	48	50	50	50	58	58	65
11	14	29	31	35	36	41	49	55	55	58	59	61	62	67
15	20	30	32	39	42	42	50	59	60	60	60	65	68	71
16	21	35	41	41	43	44	58	62	69	69	70	70	70	75
20	22	36	41	42	50	50	61	65	70	75	75	76	78	78
21	25	37	44	44	59	60	62	70	72	75	76	78	78	80
22	28	39	48	50	61	62	66	71	75	75	76	78	81	85
26	31	40	56	65	65	65	69	75	78	78	80	82	82	88
29	34	41	61	66	69	72	72	78	80	80	81	82	84	88
31	41	45	66	67	70	72	72	78	82	84	84	85	85	91
32	45	49	67	67	72	78	80	81	86	85	86	86	88	95
40	55	56	70	71	75	81	81	81	86	86	88	91	93	98

Figure 1.1.3: Algorithm 3 Grid Search

Algorithm 4 Searching in a Sorted Grid

- Search *X* starting with the digit at bottom left corner *(say Y)*
- Recursively perform below steps until *X* is found:
    - *Y = X:* number found
    - *Y* $<$ *X:* skip the column above, move to the right
    - *Y* $>$ *X:* skip the row to the right, move in column above
- Let $Q_4(n)$ = number of comparisons in the $n \times n$ grid
- Then in the worst case scenario the number of comparisons are 2n - 1: $Q_4(n) \leq 2n - 1 < 2n$



2	5	10	11	20	22	26	30	31	31	34	34	37	40	45
4	6	15	18	27	30	31	38	39	40	42	42	44	48	48
7	9	16	21	27	31	39	41	41	41	48	50	55	55	59
8	13	22	22	27	34	40	45	48	50	50	50	58	58	65
11	14	29	31	35	36	41	49	55	55	58	59	61	62	67
15	20	30	32	39	42	42	50	59	60	60	60	65	68	71
16	21	35	41	41	43	44	58	62	69	69	70	70	70	75
20	22	36	41	42	50	50	61	65	70	75	75	76	78	78
21	25	37	44	44	59	60	62	70	72	75	76	78	78	80
22	28	39	48	50	61	62	66	71	75	75	76	78	81	85
26	31	40	56	65	65	65	69	75	78	78	80	82	82	88
29	34	41	61	66	69	72	72	78	80	80	81	82	84	88
31	41	45	66	67	70	72	72	78	82	84	84	85	85	91
32	45	49	67	67	72	78	80	81	86	85	86	86	88	95
40	55	56	70	71	75	81	81	81	86	86	88	91	93	98

Figure 1.1.4: Algorithm 4 Grid Search

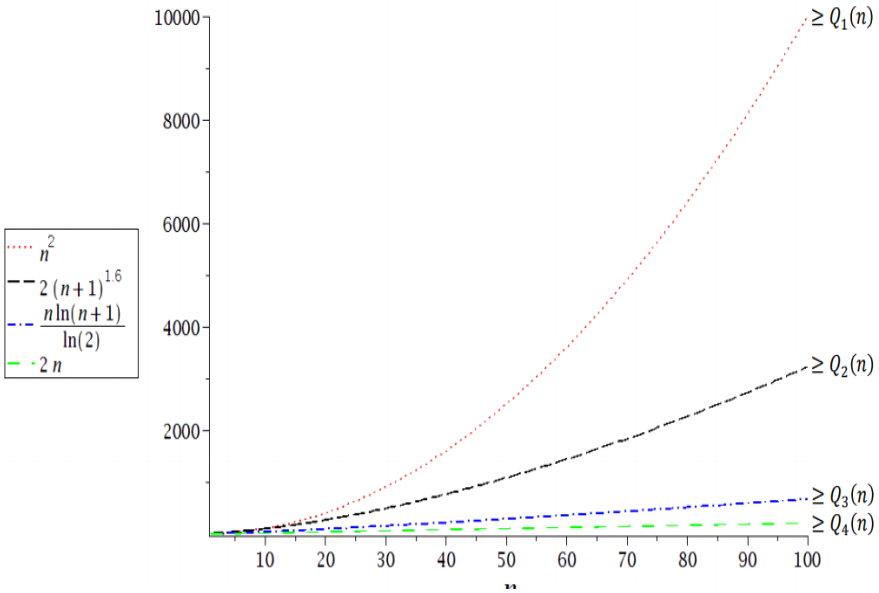
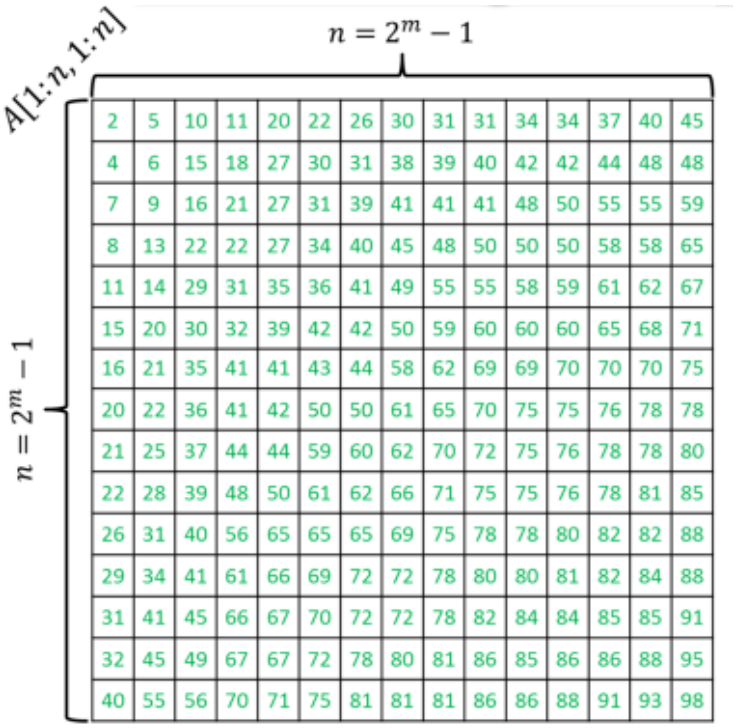


Figure 1.1.5: Comparison Graph of four Grid Search Algorithms

Comparison of the four algorithmic approaches shows that algorithm four is the most optimal algorithm in terms of number of comparisons required

1.2 Calculating running time of an algorithm

Number comparison grid



2	5	10	11	20	22	26	30	31	31	34	34	37	40	45
4	6	15	18	27	30	31	38	39	40	42	42	44	48	48
7	9	16	21	27	31	39	41	41	41	48	50	55	55	59
8	13	22	22	27	34	40	45	48	50	50	50	58	58	65
11	14	29	31	35	36	41	49	55	55	58	59	61	62	67
15	20	30	32	39	42	42	50	59	60	60	60	65	68	71
16	21	35	41	41	43	44	58	62	69	69	70	70	70	75
20	22	36	41	42	50	50	61	65	70	75	75	76	78	78
21	25	37	44	44	59	60	62	70	72	75	76	78	78	80
22	28	39	48	50	61	62	66	71	75	75	76	78	81	85
26	31	40	56	65	65	65	69	75	78	78	80	82	82	88
29	34	41	61	66	69	72	72	78	80	80	81	82	84	88
31	41	45	66	67	70	72	72	78	82	84	84	85	85	91
32	45	49	67	67	72	78	80	81	86	85	86	86	88	95
40	55	56	70	71	75	81	81	81	86	86	88	91	93	98

Figure 1.2.6: Grid to find X

1: **for** $i = 1$ to n **do**
2: **for** $j = 1$ to n **do**
3: if $A[i, j] = x$ then return ”item found”
4: **end for**
5: **end for**
6: return ”item not found”

Although we could compute the worst case bound for the algorithm, we can not guarantee that it is the same as running time, as running time depends on a variety of factors like the expertise of the programmer , the configuration of the machine used to execute it, the way it is written and the compiler used. In the above algorithm, line 1 executes n times and line 2 executes n^2 times in the worst case scenario. Line 6 executes only once If $T_1(n)$ is the running time of the algorithm, then $T_1(n) \leq a_1n^2 + a_2n + a_3$, where *a1, a2 and a3* are constants. For large enough n, we can ignore the lower order terms and assuming

$n \geq a2 + a3$, $T_1(n) \leq (a1+1)n^2 = (a1+1)Q_1(n)$

Table 1: Running times of the Four Algorithms

GRID SEARCHING ALGORITHM	**WORST-CASE BOUND ON COMPARISONS**	**WORST-CASE BOUND ON RUNNING TIMES**
ALGORITHM 1	$Q_1(n) \leq n^2$	$T_1(n) \leq c_1 n^2$
ALGORITHM 2	$Q_2(n) \leq 2(n+1)^{1.6}$	$T_2(n) \leq c_2(n+1)^{1.6}$
ALGORITHM 3	$Q_3(n) \leq nlog_2(n+1)$	$T_3(n) \leq c_3 nlog_2(n+1)$
ALGORITHM 4	$Q_4(n) \leq 2n$	$T_4(n) \leq c_4 n$

1.3 Asymptotic Bounds

If running time of an algorithm is O(n^2), it means that for large enough value of n there is a constant c, such that cn^2 will be larger than f(n). Even though g(n) would be smaller than f(n) for small values of c, there will be some constant value at which it will cross the value of f(n) and stay that way.

Asymptotic Upper Bound (O - notation)

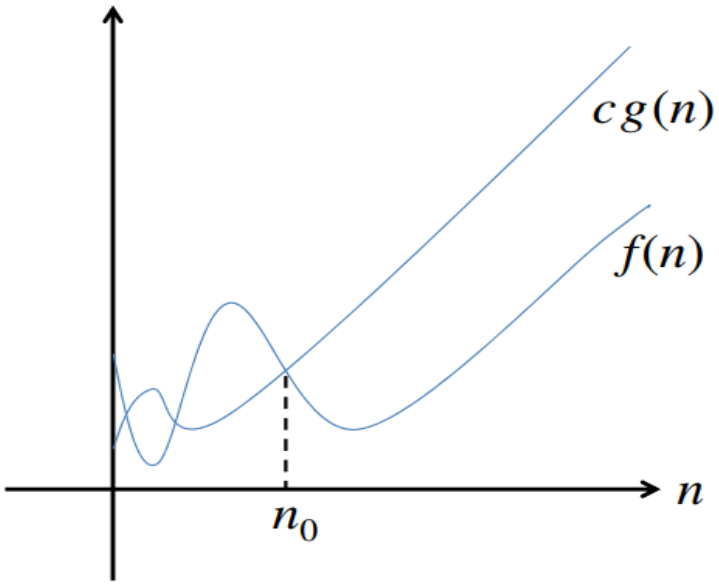


Figure 1.3.7: Big-oh notation

$$O(g(n)) = \left\{ \begin{array}{c} \text{f(n): there exist positive constants c and } n_0 \text{ such that} \\ 0 \leq f(n) \leq cg(n) \text{ for all n} \geq n_0 \end{array} \right\}$$

$$O(g(n)) = \left\{ \begin{array}{c} \text{f(n): there exists a positive constant c such that} \\ \lim_{x \to +\infty} \left(\frac{f(n)}{g(n)} \right) \leq c \end{array} \right\}$$

Asymptotic Upper Bound (O - notation)

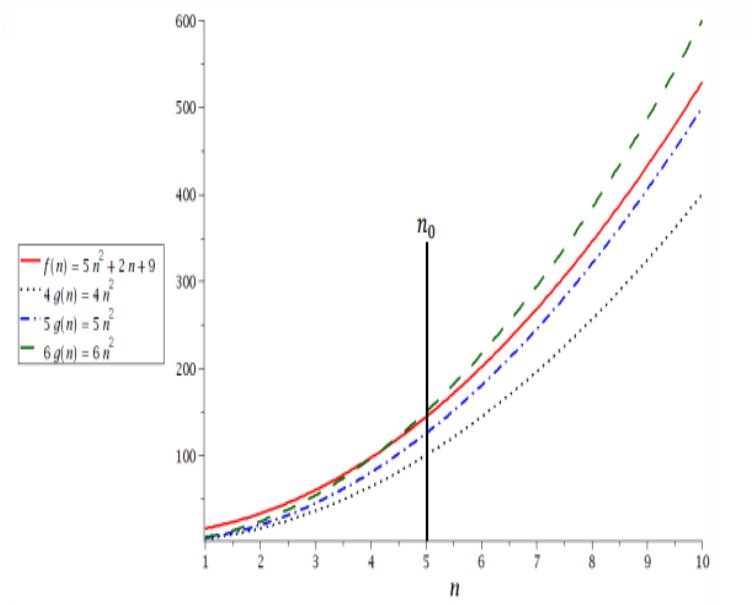


Figure 1.3.8: Big-oh notation

Let g(n) = n^2 Then f(n) = O(n^2) because: $0 \leq f(n) \leq cg(n) for c = 6 and n \geq 5$

Asymptotic Lower Bound (Ω - notation)

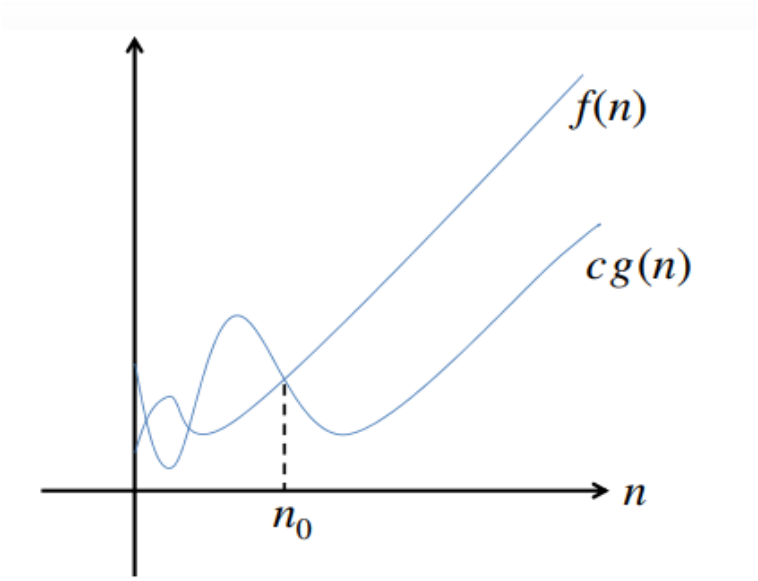


Figure 1.3.9: Big-omega notation

$$\Omega(g(n)) = \left\{ \begin{array}{c} \text{f(n): there exist positive constants c and } n_0 \text{ such that} \\ 0 \leq cg(n) \leq f(n) \text{ for all n} \geq n_0 \end{array} \right\}$$

$$\Omega(g(n)) = \left\{ \begin{array}{c} \text{f(n): there exists a positive constant c such that} \\ \lim_{x \to +\infty} \left(\frac{f(n)}{g(n)} \right) \geq c \end{array} \right\}$$

Asymptotic Lower Bound (Ω - notation)

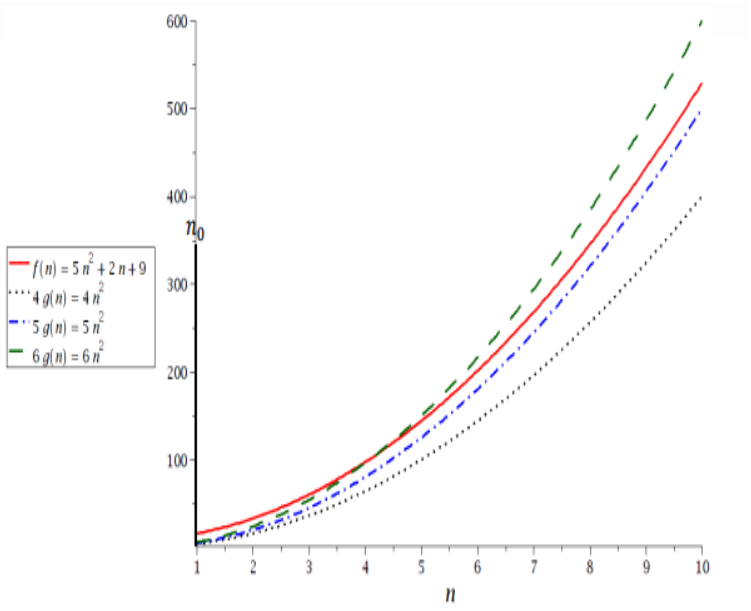


Figure 1.3.10: Big-omega notation

Let g(n) = n^2. Then f(n) = Ω(n^2) because: $0 \leq cg(n) \leq f(n) for c = 5 \; and \; n \geq 1$