2.1)

- $\frac{2}{N}$ has infinite growth rate when $N=0$ and tends to zero growth rate as $N=\infty$.
- 37 is costant and independent of N.
- N is linear and grows slowly.
- The power functions: $\sqrt{N}=N^{0.5}, N^{1.5}, N^2, N^3$.
- The logarithmic functions: $N\log\log N$, $N\log N$, $N\log(N^2)$, $N\log^2 N$, $N^2\log N$.
- The exponential functions: $2^{N/2}$, 2^N.

We take a few of the above. Let's test them.

- $N\log N$ and $N\log(N^2)$:

  $N\log(N^2) = N(2\log N) = 2N\log N = O(N\log N)$

  the <u>same growth rate</u> as $N\log N$ which is $O(N\log N)$.

- $2^{N/2}$ and 2^N:

  $$\lim_{N\to\infty}\frac{2^{N/2}}{2^N} = \lim_{N\to\infty}\frac{2^{N/2}}{2^{N/2}\times 2^{N/2}} = \lim_{N\to\infty}\frac{1}{2^{N/2}} = 0$$

  $2^{N/2}$ has <u>smaller</u> order of growth than 2^N.

- $N^{1.5}$ and $N\log^2 N$:

  $$\lim_{N\to\infty}\frac{N^{1.5}}{N\log^2 N} = \lim_{N\to\infty}\frac{N^{1.5-1}}{\log^2 N} = \lim_{N\to\infty}\frac{N^{0.5}}{2\log N} = \lim_{N\to\infty}\frac{0.5N^{0.5}}{\log N} = \infty$$

  $N^{1.5}$ has <u>larger</u> order of growth than $N\log^2 N$.

- $N\log^2 N$ and $N\log\log N$:

  $N\log^2 N = N(\log N)^2 = N\log N\log N \neq N\log\log N$

  thus $N\log^2 N$ and $N\log\log N$ do <u>not</u> growth at the same rate.

2.5) If there exists two costant values $c>0$ and $n>0$ such that $f(N) \leq c \times g(N)$ where $N \geq n$, then we can say that $f(N) = O(g(N))$.

or

If there exists two costant values $c>0$ and $n>0$ such that $g(N) \leq c \times f(N)$ where $N \geq n$, then we can say that $g(N) = O(f(N))$.

Consider the following functions:

$f(N) = \cos(N)$

$g(N) = \sin(N)$

To say that $f(N) = O(g(N))$ is not true, we need to prove that $f(N) \leq c \times g(N)$ is not possible.

Now, assume that N is from 0 to n, where $n>0$.

N (degree term)	0	30	45	60	90
$g(N) = \sin(N)$	0	0.50	0.707	0.866	1
$f(N) = \cos(N)$	1	0.866	0.707	0.50	0

$g(N) < f(N)$ $\quad$ $g(N) = f(N)$ $\quad$ $g(N) > f(N)$

From the above table: we conclude that $f(N) > g(N)$ and $f(N) < g(N)$.

Thus, $f(N) \leq c \times g(N)$ and $g(N) \leq c \times f(N)$ is not possible for all N values.

Result: $f(N) = O(g(N))$ and $g(N) = O(f(N))$ is <u>not true</u>.

2.7) (1) • first line executes c numer of times.
• second line : n times. • third line c inside loop
So, $n \times c = cn$. Total: $c + cn = O(N)$

(2) • first line c times.
• second line n times.
• third line n times.
• fourth line $c \times n = c.n$.
So, the total is: $c + c.n^2 = O(N^2)$

(3) • c times
• n times
• $n \times n$ times
• c times inside the loops.
Total: $c + c.n^3 = O(N^3)$

(4) The inner loop executes "i" times, the end being $(n-1)$. We have $n(n-1)$. In the worst case: $O(N^2)$ based on the previous analysis.

(5) "j" can be as large as "i^2", which could be as large as N^2. "k" can be as large as "j", which is N^2. The total is: $N.N^2.N^2 = O(N^5)$

(6) If we ignore the outer loop for now and analyze the problem in terms of i.
The mid loop runs i^2 times. The inner loop is invoked whenever $j \% i == 0$, that means: $i, 2i, 3i, \ldots, i^2$. At each time we run until the relevant j, so this means:

$i + 2i + 3i + \dots + (i-1) \cdot i = i(1 + 2 + \dots + i-1) =$

$= i \cdot (i \cdot (i-1)/2)$

The last equality comes from the sum of arithmetic progression.

The above is $O(i^3)$.

If we repeat for the outer loop which is 1 to n, we have $O(N^4)$ in total.

2.11) a) $N_1 = 100$, $T(N_1) = 0.5\,ms$, $N_2 = 500$

$$T(N_2) = \frac{N_2}{N_1} \times T(N_1)$$

$$= \frac{500}{100} \times 0.5$$

$$= 5 \times 0.5$$

$$= 2.5\,ms$$

b)

$$T(N_2) = \frac{N_2 \log N_2}{N_1 \log N_1} \times T(N_1)$$

$$= \frac{500 \times \log_2 500}{100 \times \log_2 100} \times 0.5$$

$$= 5 \times \frac{\log_2 500}{\log_2 100} \times 0.5$$

$$= 2.5 \times \frac{\log_2 500}{\log_2 10^2}$$

$$T(N_2) = 2.5 \times \frac{\log_2 500}{2 \times \log_2 10}$$

Since, $\frac{\log_c(a)}{\log_c(b)} = \log_b a$

$$= \frac{2.5}{2} \times \log_{10} 500$$

$$= 1.25 \times \log_{10} 500$$

$$= 1.25 \times 2.698$$

$$= 3.3725\,ms$$

c) $T(N_2) = \dfrac{N_2^2}{N_1^2} \times T(N_1)$

$= \dfrac{500^2}{100^2} \times 0.5$

$= 25 \times 0.5$

$= 12.5\ ms$

d) $T(N_2) = \dfrac{N_2^3}{N_1^3} \times T(N_1)$

$= \dfrac{500^3}{100^3} \times 0.5$

$= 125 \times 0.5$

$= 62.5\ ms$

2.12) $N_1 = 100$, $T(N_1) = 0.5ms$, $T(N_2) = 1 min$

a) $T(N_2) = 1 min = 60 sec = 60{,}000 ms$

$$N_2 = \frac{T(N_2)}{T(N_1)} \times N_1$$

$$= \frac{60000}{0.5} \times 100$$

$$= 120000 \times 100$$

$$= 12 \times 10^6 \text{ input size}$$

b) $$N_2 \log N_2 = \frac{T(N_2)}{T(N_1)} \times N_1 \log N_1$$

$$= \frac{60000}{0.5} \times 100 \times \log_2 100$$

$$= 120\,000 \times 100 \times \log_2 100$$

$$= 12 \times 10^6 \times 6.643856$$

$$= 79.726272 \times 10^6$$

$$N_2 \log N_2 = 79.726272 \times 10^6$$

$$\log_2 N_2^{N_2} = 79.726272 \times 10^6$$

$$N_2^{N_2} = 2^{79.726272 \times 10^6}$$

$$N_2 \simeq 3.6522 \times 10^6$$ input size

Wolfram Mathematica
or
Newton's method

c) $N_2^2 = \frac{T(N_2)}{T(N_1)} \times N_1^2$

$= \frac{60000}{0.5} \times (100)^2$

$= 120000 \times (100)^2$

$= 12 \times 10^4 \times 10^4$

$N_2 = \sqrt{12 \times 10^4 \times 10^4}$

$= 3.46410 \times 10^4$

$= 34\,641$ input size

d) $N_2^3 = \frac{T(N_2)}{T(N_1)} \times N_1^3$

$= \frac{60000}{0.5} \times (100)^3$

$= 120000 \times (100)^3$

$N_2 = \sqrt[3]{120000 \times (100)^3}$

$= 49.324 \times 100$

$= 4932$ input size

2.25) Program A: $150 N\log_2 N$ (1)
Program B: N^2 (2)

a) Consider N to be the large power of 10, let's say 10^6.

(1) $\overset{10^6}{\Rightarrow}$ $150 N \log_2 N = 150 \times 10^6 \times \log_2 10^6$

$= 3 \times 10^9$ (3)

(2) $\overset{10^6}{\Rightarrow}$ $N^2 = (10^6)^2 = 10^{12}$ (4)

(3) < (4), so A runs faster than B.

b) Consider N to be the small value power of 10, let's say 10.

(1) $\overset{10}{\Rightarrow}$ $150 N \log_2 N = 150 \times 10 \times \log_2 10$

$= 4950$ (5)

(2) $\overset{10}{\Rightarrow}$ $N^2 = (10^1)^2 = 10^2$ (6)

(6) < (5), so B runs faster than A.

c) Besides the worst case, the running time also depends on the following information:
- hardware configuration
  - storage
  - clock speed
  - the bit operations of a program

It is difficult to determine based on average values.

d) Yes, according to (c).

However, Program A seems to run faster for large inputs and vice versa (for small inputs). See a) and b). Usually, we are considered about large inputs in most algorithms, where B <u>cannot</u> run faster than A.

3.1) Check code (on Moodle).

- The for loop run from $i=0$ to size of the list P.
- The total number of iterations is equal to the size of the list P.
- Running time is $O(N)$, where n is the size of the list P.

3.7)

- The makeList method creates an object for the ArrayList to store the integers.
- It adds the integers 1 through given N at the end of the list and trims to the size of the list.
- It uses a for loop and adds i. Then, uses trimToSize

- The for loop executes N times, So, it needs $O(N)$.
- The add is costant, so $O(1)$.
- The trimToSize() executes N times to move the elements from the old list to the new list. So, it needs $O(N)$.
- The total time is: $N \times (c+N) = cN + N^2$
- For the makeList method we have:

$$O(N) + O(N^2) = O(N^2)$$

3.8)

- The removeFirstHalf method accepts a list of elements as a parameter.
- It removes the first half of the elements from the list.
- It uses ~~the size()~~ the size and remove methods for the list and a for loop.

---

a) In each iteration of the for loop, the size method returns a different value, which is 1 less than its previous value. It causes to produce the incorrect result. If we always call the size() in the loop, the running time will increase.

b)
- The size will take (c) costant time.
- The call to remove method needs linear time.
- The for loop calls the removeFirstHalf N times.
- The total for the for loop block is: $N \times N = N^2$.
- For the entire method we have:

$$N^2 + c = O(N^2) + O(1) = O(N^2)$$

3.8) c)
Continuation →

- The size neends (c) time.
- The remove method takes (c) costant time. Since there is no need to loop.
- The for loop is again linear so N times.
- For the for loop block we have: $N \times c = c \cdot N$
- The total will be:

$$cN + c = O(N) + O(1) = O(N)$$

d)

```
java.util.Iterator itr= lst.iterator()
int i=0;
while (itr.hasNext() && i<theSize)
{
 itr.remove();
 i++;
}
```

For both b) and c) data structures ~~nothing~~ will change in terms of time complexity.

3.20) a) Lazy deletion:

Advantages

- Lazy deletion takes less time to delete the node as it only marks the node as deleted rather than removing it from the list. Therefore, it takes less time as a normal deletion.
- It makes programming simpler if the operation performed on the list is not only deletion.

Disadvantages

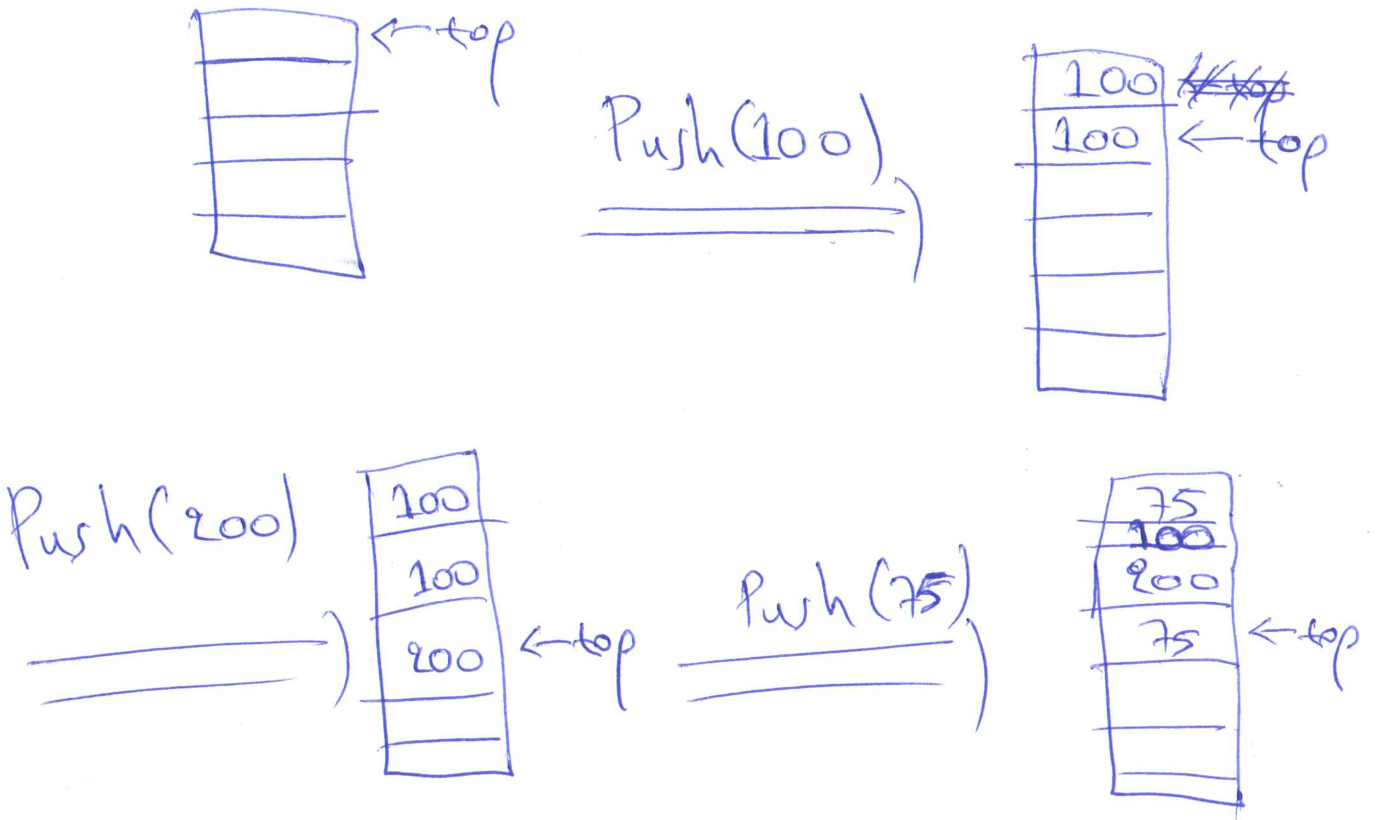
- The list takes extra unnecessary space. After performing a deletion, the node which is been deleted occupy the space in the disc.
- Since the deleted nodes are only marked, it takes more time to traverse the list.

Search the web and check Moodle for more.

3.25) a) 1st option

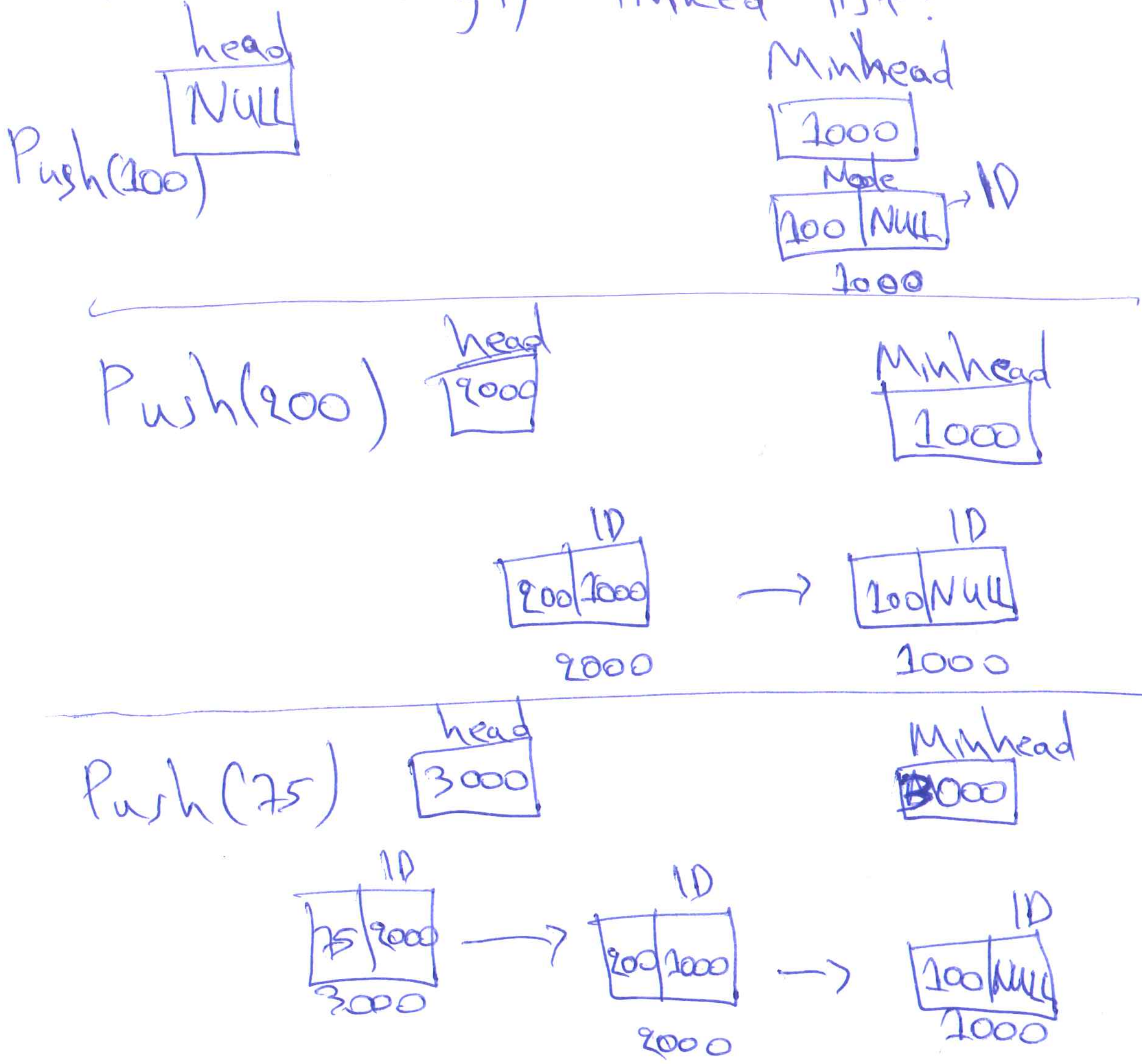
Use two stacks

- Push and pop the items associated with the S1 (stack 1), compared with the top of S2.
- If the item that is pushed is less then push a copy on the S2.
- If the contents of the S1 are popped and the result is equal then pop the contents of S2.



2nd option

For the singly linked list:



b) Check Moodle.

3.28) Check the code on Moodle.