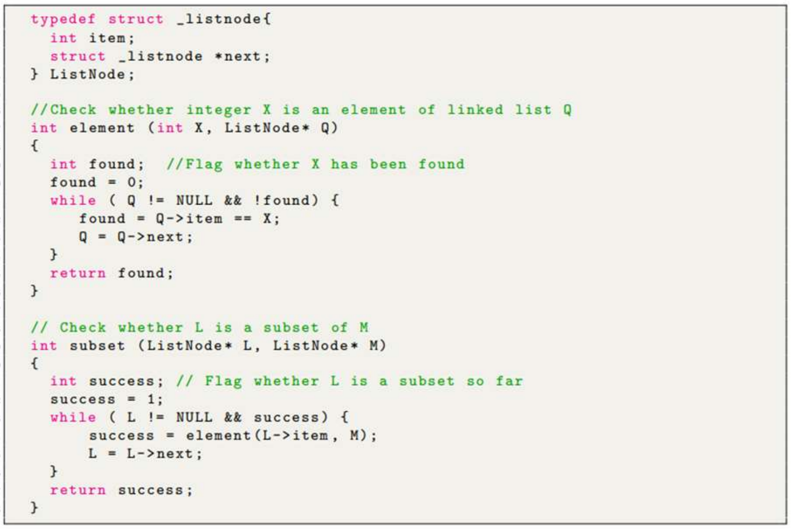
Question 1

- The function subset() takes two linked lists of integers and determines whether the first is a subset of the second.
- Give the worst-case running time of subset as a function of the lengths of the two lists.
- When will this worst case happen?

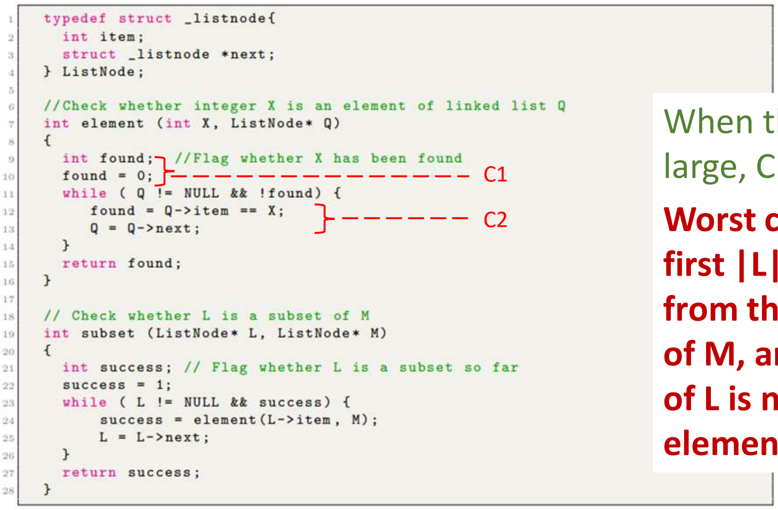


```c
typedef struct _listnode{
  int item;
  struct _listnode *next;
} ListNode;

//Check whether integer X is an element of linked list Q
int element (int X, ListNode* Q)
{
  int found;  //Flag whether X has been found
  found = 0;
  while ( Q != NULL && !found) {
     found = Q->item == X;
     Q = Q->next;
  }
  return found;
}

// Check whether L is a subset of M
int subset (ListNode* L, ListNode* M)
{
  int success; // Flag whether L is a subset so far
  success = 1;
  while ( L != NULL && success) {
      success = element(L->item, M);
      L = L->next;
  }
  return success;
}
```

Question 1

M

L



```c
typedef struct _listnode{
  int item;
  struct _listnode *next;
} ListNode;

//Check whether integer X is an element of linked list Q
int element (int X, ListNode* Q)
{
  int found;  //Flag whether X has been found
  found = 0;
  while ( Q != NULL && !found) {
     found = Q->item == X;
     Q = Q->next;
  }
  return found;
}

// Check whether L is a subset of M
int subset (ListNode* L, ListNode* M)
{
  int success; // Flag whether L is a subset so far
  success = 1;
  while ( L != NULL && success) {
      success = element(L->item, M);
      L = L->next;
  }
  return success;
}
```

When the size of M is large, C1 is negligible.

Worst case scenario: the first |L|-1 elements of L are from the last |L|-1 elements of M, and (the last element of L is not in M, or is the last element of M).

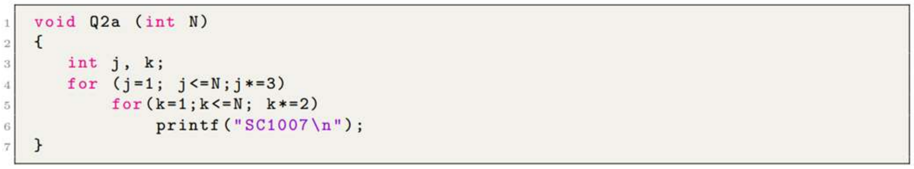
The running time:

$$
\begin{aligned}
&= |\boldsymbol{M}| + (|\boldsymbol{M}| - \mathbf{1}) + \cdots + (|\boldsymbol{M}| - (|\boldsymbol{L}| - \mathbf{1}))\\
&= |\mathbf{L}||\boldsymbol{M}| - (\mathbf{1} + \mathbf{2} + \cdots + (|\boldsymbol{L}| - \mathbf{1}))\\
&= |\boldsymbol{L}||\boldsymbol{M}| - \frac{(|\boldsymbol{L}|)(|\boldsymbol{L}| - \mathbf{1})}{\mathbf{2}}\\
&= \Theta|\boldsymbol{L}||\boldsymbol{M}|
\end{aligned}
$$

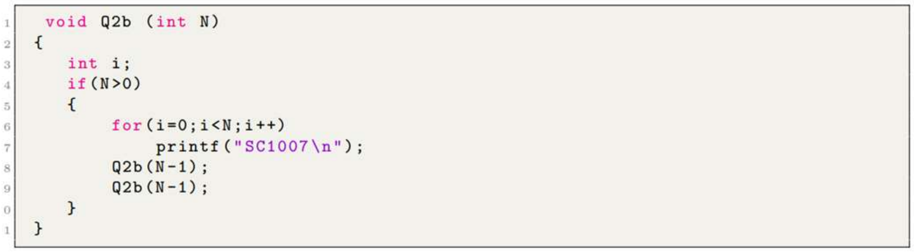
$$\boxed{1 + 2 + \cdots + n = \frac{(1+n)n}{2}}$$

Question 2

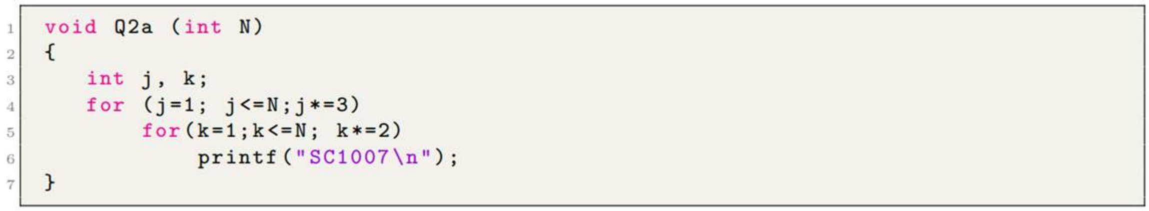
- Find the number of printf used in the following functions. Write down its time complexity in Θ notation in terms of *N*.



```
void Q2a (int N)
{
    int j, k;
    for (j=1; j<=N;j*=3)
        for(k=1;k<=N; k*=2)
            printf("SC1007\n");
}
```



```
 void Q2b (int N)
{
    int i;
    if(N>0)
    {
        for(i=0;i<N;i++)
            printf("SC1007\n");
        Q2b(N-1);
        Q2b(N-1);
    }
}
```



```c
void Q2a (int N)
{
   int j, k;
   for (j=1; j<=N;j*=3)
       for(k=1;k<=N; k*=2)
           printf("SC1007\n");
}
```

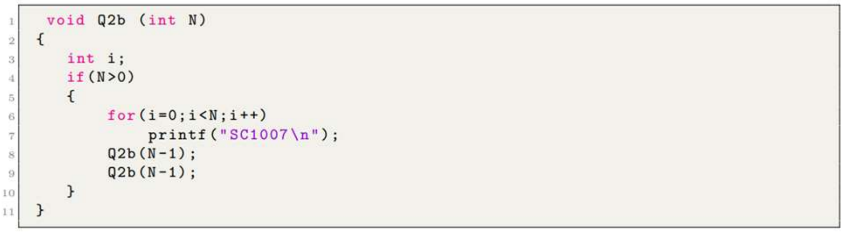
- For the inner loop:

$$2^{K-1} \leq N \leq 2^{K}$$
$$(K-1) \leq log_2 N \leq \mathrm{K}$$
$$K \leq log_2 N + 1 \leq K + 1$$
$$K = \lfloor log_2 N \rfloor + 1$$

- For the outer loop:

$$3^{J-1} \leq N \leq 3^{J}$$
$$(J-1) \leq log_3 N \leq \mathrm{J}$$
$$J \leq log_3 N + 1 \leq J + 1$$
$$J = \lfloor log_3 N \rfloor + 1$$

- The number of printf is $JK = (\lfloor log_3 N \rfloor + 1)(\lfloor log_2 N \rfloor + 1)$
- The time complexity is $\Theta\left(\left(log_2 N\right)^2\right)$ $(as\ N \rightarrow infinity)$



```c
void Q2b (int N)
{
   int i;
   if(N>0)
   {
       for(i=0;i<N;i++)
           printf("SC1007\n");
       Q2b(N-1);
       Q2b(N-1);
   }
}
```

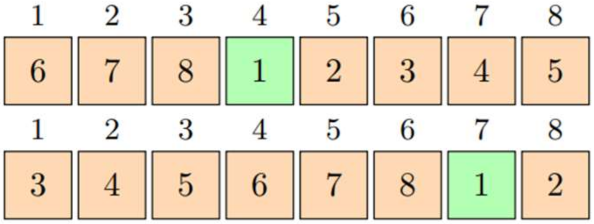
- $W_1 = 1$
- $W_N = N + W_{N-1} + W_{N-1}$

$$= N + 2W_{N-1}$$
$$= N + 2(N-1+2W_{N-2})$$
$$= N + 2(N-1) + 2^2 W_{N-2}$$
$$= \mathrm{N} + 2(\mathrm{N}-1) + 2^2(N-2) + \cdots + 2^{N-1}(1) = \sum_{t=0}^{N-1} 2^t (N-t)$$

- The time complexity is $\Theta(2^N)$

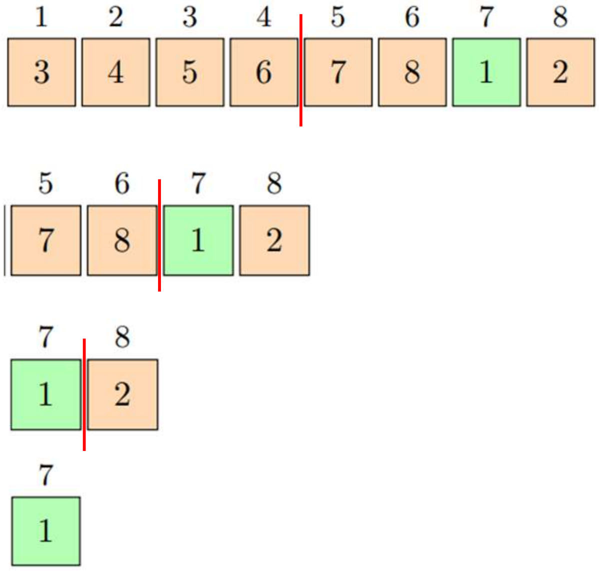
Question 3

- A sequence, $x_1, x_2, \ldots, x_n$, is said to be cyclically sorted if the smallest number in the sequence is x_i $for\ some\ i$, and the sequence, $x_i, x_{i+1}, \ldots, x_n, x_1, x_2, \ldots, x_{i-1}$ is sorted in increasing order. Design an algorithm to find the minimal element in the sequence in $O(logn)$ time. What is the worst-case scenario?



1	2	3	4	5	6	7	8
6	7	8	1	2	3	4	5

1	2	3	4	5	6	7	8
3	4	5	6	7	8	1	2



1	2	3	4	5	6	7	8
3	4	5	6	7	8	1	2

middle = 6, middle > last, i.e., 2.
The minimum is in the second half

5	6	7	8
7	8	1	2

middle = 8, middle > last, i.e., 2
The minimum is in the second half

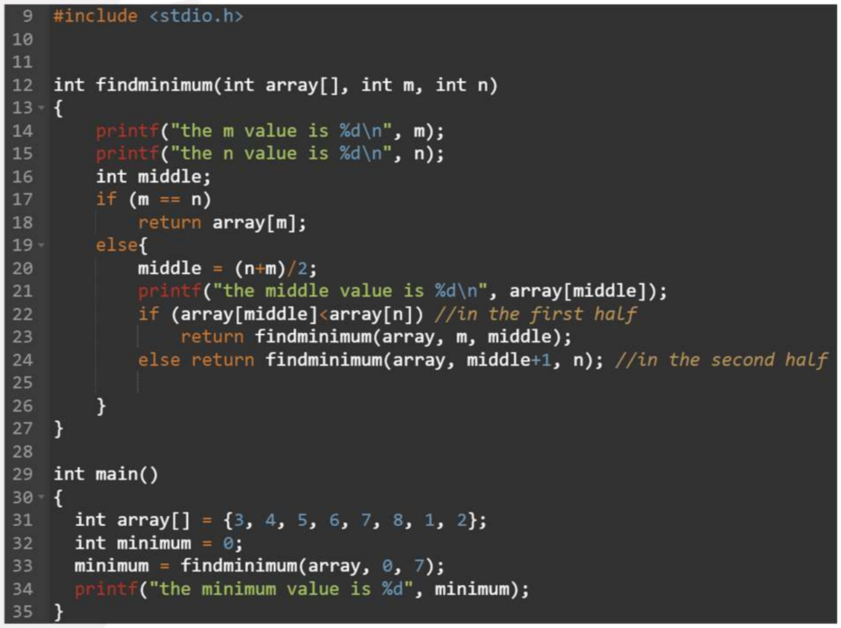
7	8
1	2

middle = 1, middle<last, i.e., 2
The minimum is in the first half

7
1

Only one element

- The number of comparisons
  - W1 = 1
  - W2 = 2
  - W4 = 3
  - W8 = 4
  - ….
- Time complexity
  - $\mathrm{T}(n) = T\left(\frac{n}{2}\right) + c$
- Worst Case scenario
  - We need to cut the array until only one element is left.
  - No differences among scenarios.



```c
#include <stdio.h>


int findminimum(int array[], int m, int n)
{
    printf("the m value is %d\n", m);
    printf("the n value is %d\n", n);
    int middle;
    if (m == n)
        return array[m];
    else{
        middle = (n+m)/2;
        printf("the middle value is %d\n", array[middle]);
        if (array[middle]<array[n]) //in the first half
            return findminimum(array, m, middle);
        else return findminimum(array, middle+1, n); //in the second half

    }
}

int main()
{
  int array[] = {3, 4, 5, 6, 7, 8, 1, 2};
  int minimum = 0;
  minimum = findminimum(array, 0, 7);
  printf("the minimum value is %d", minimum);
}
```