```
i = 1;
while(i<n){
  Statements;
  i=i*2;
}
```

i
1
2
2*2
2*2*2
2*2*2*2
.
.
.
2*2*2*2*2. . . . k times

terminates

i >=n

2^k>=n Since i is

log 2^k>=log n

k>=log n

k= log n

So it's O(log n)

the loops i is incrementing by 2 each time.

```
or(i=0;i<n;i=i*2){
Statements;
```

```
getTimeComplexity(int n){
   if(n<5){
       print("n less than 5"+n);     →  1
   }
   else{
       for(i=0;i<n;i++){             →  n+1
           print(i);                 →  1
```

n<5 Best case

Time complexity= O(1)

n>=5 Worst case

Time complexity = (n+1)1

= n+1

= n

```
vector<int> func(vector<int> &g1, int k, vector<int> step)
{
    int curr, count;
    int n = step.size();
    vector<int> intermed;
    for(int i = 0; i < n; i++)
    {
        count = 1;
        for(int j = i + 1; j < n; j++)
        {
            if(step[i] == step[j])
            {
                curr = step[i];
                count ++;
            }
            if(step[i] != step[j])
            {
                curr = step[i];
                intermed.push_back(count);
                intermed.push_back(curr);
                i = j;
                curr = step[j];
                count = 1;
            }
        }
        if(i = n-1)
        {
            intermed.push_back(count);
            intermed.push_back(curr);
        }
    }
    g1.insert(g1.end(), intermed.begin(), intermed.end());
    return intermed;
}
```

```
i =0;           →  1
while(i<n){     →  n+1
  statements;   →  1
  i++;          →  1
}
```

Total time comlexity

= 1+(n+1)(1+1)

= 1+(n+1)(2)

= 1+2n+2

= 2n+3

= 2n

```
for(i =0;i<n;i++){   →  n+1
  statements;        →  1
}
```

Total time complexity

= (n+1)1

= n+1

=n

So its O(n)

```
void retry(int k, vector<int> &g1, vector<int> step) {
    try
    {
        g1.at(k);

    } catch (const out_of_range& oor)
    {
        step = func(g1, k, step);
        retry(k, g1, step);
    }
}

int kthDigit(int k)
{
    vector<int> step = {1,0};
    vector<int> g1 = {0,1,0};
    retry(k, g1, step);
    return g1.at(k-1);
}
```

$T(1) = 4$

Build Solution

$$T(n) = 2T(n/2) + 4n$$
$$= 2\left[2T\left(\frac{n}{2^2}\right) + 4\frac{n}{2}\right] + 4n$$
$$= 2^2 T\left(\frac{n}{2^2}\right) + 4n + 4n$$
$$= 2^2\left[2T\left(\frac{n}{2^3}\right) + 4\frac{n}{2^2}\right] + 4n + 4n$$
$$= 2^3 T\left(\frac{n}{2^3}\right) + 4n + 4n + 4n$$
$$= 2^3\left[2T\left(\frac{n}{2^4}\right) + 4\frac{n}{2^3}\right] + 4n + 4n + 4n$$
$$= 2^4 T\left(\frac{n}{2^4}\right) + \underbrace{4n + 4n + 4n + 4n}_{4}$$
$$= 2^i T\left(\frac{n}{2^i}\right) + i(4n)$$
$$\frac{n}{2^i} = 1 \Rightarrow n = 2^i \Rightarrow \log_2 n = i$$

Expand Scratch

$$T(n/2) = 2T\left(\frac{n}{2^2}\right) + 4\left(\frac{n}{2}\right)$$
$$T\left(\frac{n}{2^2}\right) = 2T\left(\frac{n}{2^3}\right) + 4\left(\frac{n}{2^2}\right)$$
$$T\left(\frac{n}{2^3}\right) = 2T\left(\frac{n}{2^4}\right) + 4\frac{n}{2^3}$$

$$2^{\log_2 n} = n^{\log_2 2} = n$$

$$\Rightarrow 2^{\log_2 n} T(1) + (\log_2 n)(4n)$$
$$= n(4) + 4n\log_2 n$$
$$= 4n + 4n\log_2 n = O(n\log n)$$

t

```cpp
int sumProductDigit(int a, int b)
{
    int ans = a*b;
    if(ans == 0)
    {
        return 0;
    }
    else if(ans % 9 == 0)
    {
        return 9;
    }
    else
    {
        return ans % 9;
        // the digit sum of an integer = int % 9
```

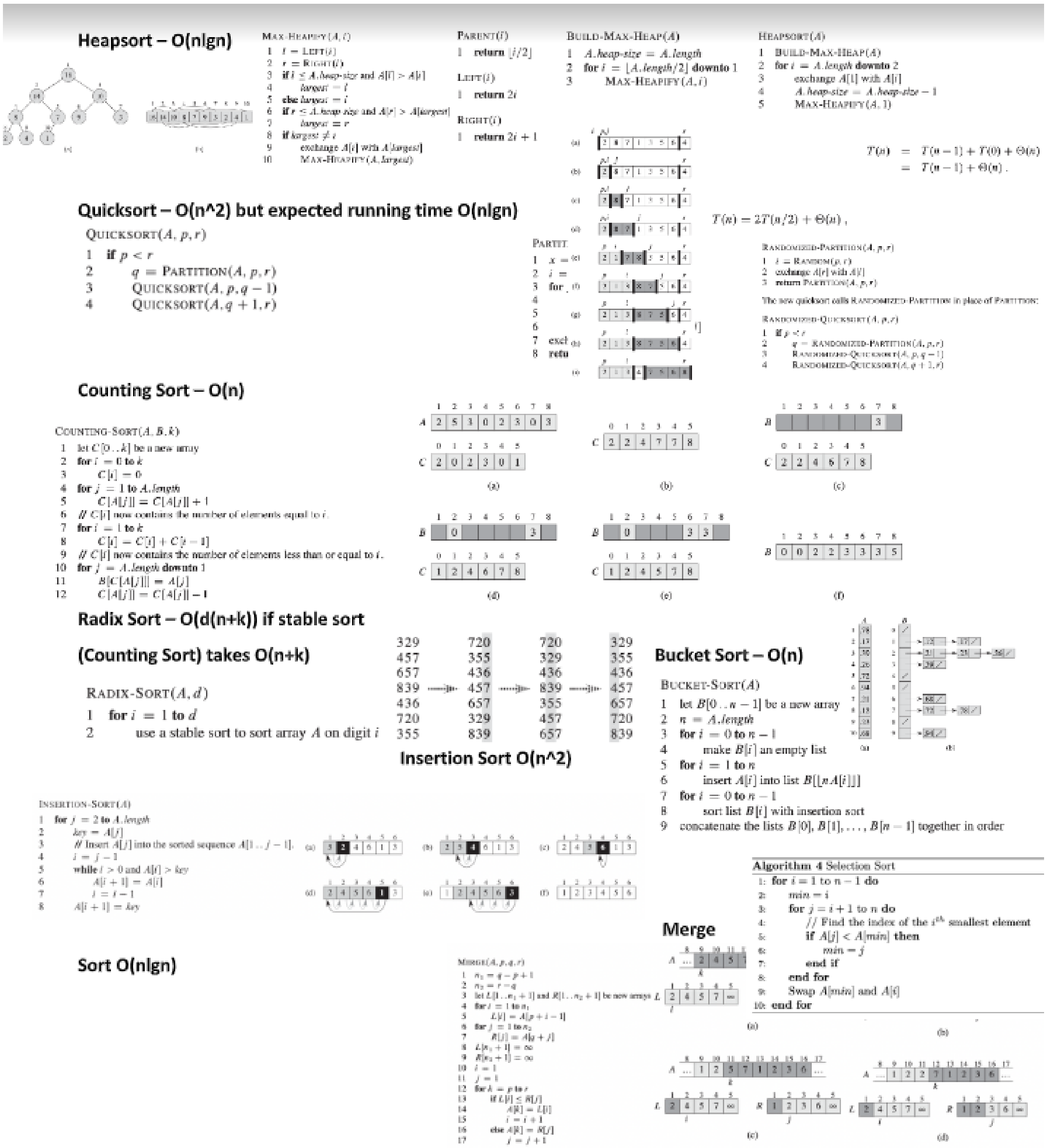
```cpp
int findKthSmallest(vector<int> A, vector<int> B, int k) {

    // Move elements from vector B to the end of vector A
    A.insert(A.end(), B.begin(), B.end());

    sort(A.begin(), A.end());

    return A[k-1]; // you will need to change this
}
```

```cpp
void zigzagSort(vector<int> &nums) {

bool flip = true;

    for (int i=0; i<=nums.size()-2; i++)
    {
        if (flip)  // allows the if else lo
        {
            if (nums[i] > nums[i+1])
                swap(nums[i], nums[i+1]);
        }
        else
        {
            if (nums[i] < nums[i+1])
                swap(nums[i], nums[i+1]);
        }
        flip = !flip; // flip the operatio
    }
}
```

```cpp
ListNode* insertionSortList(ListNode* head) {

    ListNode *start = head;

    ListNode *insert = start; // insertion ptr to insert data

    head = head->next;

    while(head != NULL){ // avoid seg fault
        insert = start;
        //head = head->next;
        while(insert != head){
            if(insert->val < head->val){
                int temp = head->val; // store head's data so it is not l
                head->val = insert->val; // move the smaller int into loc
                insert->val = temp; // move the larger int into location
            }
            else{
                insert = insert->next;
            }
        }
        head = head->next;
    }

    return start;
}
```

```cpp
ListNode* findCycleStart(ListNode* head) {
    if (head == NULL || head->next == NULL) // re
        return NULL;

    ListNode *slow = head;
    ListNode *fast = head;

    slow = slow->next;
    fast = fast->next->next; // initialize two po

    while (fast && fast->next) {
        if (slow == fast)
            break;
        slow = slow->next;
        fast = fast->next->next; // test for poin
    }

    if (slow != fast){
        return NULL;
    }

    slow = head;
    while (slow != fast) {
        slow = slow->next;
        fast = fast->next; // if loop exists, poi
    }

    return slow;
}
```

$$
\begin{aligned}
a &= b^{\log_b a}, \\
\log_c(ab) &= \log_c a + \log_c b, \\
\log_b a^n &= n \log_b a, \\
\log_b a &= \frac{\log_c a}{\log_c b}, \\
\log_b(1/a) &= -\log_b a, \\
\log_b a &= \frac{1}{\log_a b}, \\
a^{\log_b c} &= c^{\log_b a},
\end{aligned}
$$



Merge sort -- O(nlogn)q

```
STACK-EMPTY(S)
  if S.top == 0
      return TRUE
  else return FALSE
```

```
PUSH(S, x)
  S.top = S.top + 1
  S[S.top] = x
```

```
POP(S)
  if STACK-EMPTY(S)
      error "underflow"
  else S.top = S.top - 1
      return S[S.top + 1]
```

```
ENQUEUE(Q, x)
  Q[Q.tail] = x
  if Q.tail == Q.length
      Q.tail = 1
  else Q.tail = Q.tail + 1
```

```
DEQUEUE(Q)
  x = Q[Q.head]
  if Q.head == Q.length
      Q.head = 1
  else Q.head = Q.head + 1
  return x
```

```
LIST-SEARCH(L, k)
  x = L.head
  while x ≠ NIL and x.key ≠ k
      x = x.next
  return x
```

```
LIST-DELETE(L, x)
  if x.prev ≠ NIL
      x.prev.next = x.next
  else L.head = x.next
  if x.next ≠ NIL
      x.next.prev = x.prev
```

```
LIST-DELETE'(L, x)
  x.prev.next = x.next
  x.next.prev = x.prev
```

```
LIST-SEARCH'(L, k)
  x = L.nil.next
  while x ≠ L.nil and x.key ≠ k
      x = x.next
  return x
```

```
LIST-INSERT'(L, x)
  x.next = L.nil.next
  L.nil.next.prev = x
  L.nil.next = x
  x.prev = L.nil
```

$2^{2^{n+1}}$	
2^{2^n}	
$(n+1)!$	
$n!$	
$n2^n$	
e^n	
2^n	
$(\frac{3}{2})^n$	
$(\lg(n))!$	
$n^{\lg(\lg(n))}$	$\lg(n)^{\lg(n)}$
n^3	
n^2	$4^{\lg(n)}$
$n\lg(n)$	$\lg(n!)$
$2^{\lg(n)}$	n
$(\sqrt{2})^{\lg(n)}$	
$2^{\sqrt{2\lg(n)}}$	
$\lg^2(n)$	
$\ln(n)$	
$\sqrt{\lg(n)}$	
$\ln(\ln(n))$	
$2^{\lg^*(n)}$	
$\lg^*(n)$	$\lg^*(\lg(n))$
$\lg(\lg^*(n))$	
1	$n^{1/\lg(n)}$

Arithmetic series

The summation

$$\sum_{k=1}^{n} k = 1 + 2 + \cdots + n ,$$

is an ***arithmetic series*** and has the value

$$\sum_{k=1}^{n} k = \frac{1}{2}n(n+1) = \Theta(n^2) .$$

$$\sum_{k=0}^{n} k^2 = \frac{n(n+1)(2n+1)}{6} .$$

$$\sum_{k=0}^{n} k^3 = \frac{n^2(n+1)^2}{4} .$$

$$\sum_{k=0}^{n} x^k = \frac{x^{n+1}-1}{x-1} . \qquad \sum_{k=0}^{\infty} x^k = \frac{1}{1-x} .$$

$$\sum_{k=0}^{\infty} kx^k = \frac{x}{(1-x)^2}$$

for $|x| < 1$.

For positive integers n, the nth ***harmonic number*** is

$$H_n = 1 + \frac{1}{2} + \frac{1}{3} + \frac{1}{4} + \cdots + \frac{1}{n} = \sum_{k=1}^{n} \frac{1}{k} = \ln n + O(1) . \tag{A.9}$$

For any sequence $a_0, a_1, \ldots, a_n$,

$$\sum_{k=1}^{n} (a_k - a_{k-1}) = a_n - a_0 ,$$

since each of the terms $a_1, a_2, \ldots, a_{n-1}$ is added in exactly once and subtracted out exactly once. We say that the sum ***telescopes***. Similarly,

$$\sum_{k=0}^{n-1} (a_k - a_{k+1}) = a_0 - a_n .$$

it holds for n, and we prove that it holds for $n+1$

$$\sum_{k=1}^{n+1} k = \sum_{k=1}^{n} k + (n+1) = \frac{1}{2}n(n+1) + (n+1) = \frac{1}{2}(n+1)(n+2) .$$

mation. As an example, let us prove that the geometric series $\sum_{k=0}^{n} 3^k$ is $O(3^n)$. More specifically, let us prove that $\sum_{k=0}^{n} 3^k \le c3^n$ for some constant c. For the initial condition $n = 0$, we have $\sum_{k=0}^{0} 3^k = 1 \le c \cdot 1$ as long as $c \ge 1$. Assuming that the bound holds for n, let us prove that it holds for $n+1$. We have

$$\begin{aligned} \sum_{k=0}^{n+1} 3^k &= \sum_{k=0}^{n} 3^k + 3^{n+1} \\ &\le c3^n + 3^{n+1} \qquad \text{(by the inductive hypothesis)} \\ &= \left(\frac{1}{3} + \frac{1}{c}\right) c3^{n+1} \\ &\le c3^{n+1} \end{aligned}$$

as long as $(1/3 + 1/c) \le 1$ or, equivalently, $c \ge 3/2$. Thus, $\sum_{k=0}^{n} 3^k = O(3^n)$, as we wished to show

$$\frac{(k+2)/3^{k+2}}{(k+1)/3^{k+1}} = \frac{1}{3} \cdot \frac{k+2}{k+1} \le \frac{2}{3}$$

for all $k \ge 0$. Thus, we have

$$\sum_{k=1}^{\infty} \frac{k}{3^k} = \sum_{k=0}^{\infty} \frac{k+1}{3^{k+1}} \le \frac{1}{3} \cdot \frac{1}{1-2/3} = 1 .$$

$$\sum_{k=0}^{\infty} \frac{k^2}{2^k} .$$

we observe that the ratio of consecutive terms is

$$\frac{(k+1)^2/2^{k+1}}{k^2/2^k} = \frac{(k+1)^2}{2k^2} \le \frac{8}{9}$$

if $k \ge 3$. Thus, the summation can be split into

$$\begin{aligned} \sum_{k=0}^{\infty} \frac{k^2}{2^k} &= \sum_{k=0}^{2} \frac{k^2}{2^k} + \sum_{k=3}^{\infty} \frac{k^2}{2^k} \\ &\le \sum_{k=0}^{2} \frac{k^2}{2^k} + \frac{9}{8} \sum_{k=0}^{\infty} \left(\frac{8}{9}\right)^k \\ &= O(1) . \end{aligned}$$

Approximation by integrals

When a summation has the form $\sum_{k=m}^{n} f(k)$, where $f(k)$ is a monotonically increasing function, we can approximate it by integrals:

$$\int_{m-1}^{n} f(x)\,dx \le \sum_{k=m}^{n} f(k) \le \int_{m}^{n+1} f(x)\,dx . \tag{A.11}$$

When $f(k)$ is a monotonically decreasing function, we can use a similar method to provide the bounds

$$\int_{m}^{n+1} f(x)\,dx \le \sum_{k=m}^{n} f(k) \le \int_{m-1}^{n} f(x)\,dx . \tag{A.12}$$