1. $T(n) = 3T(n/2) + n$

$a = 3 \quad b = 2 \quad h(n) = n$

$\boxed{\Theta(n^{\log(3)/\log(2)})}$

2. $T(n) = T(\frac{n}{2}) + T(\frac{n}{3}) + n$

$a = 1 \quad b = 6 \quad h(n) = n$

$T(n) = 6n$

$\boxed{\Theta(n)}$

3. $T(n) = 2T(n-1) + 2$

$T(n) = (k+4)2^{n-1} - 2$

excluding constants

$\boxed{\Theta(2^n)}$

4. $T(n) = 3T(n/3) + n^2$

$a = 3 \quad b = 3 \quad h(n) = n^2$

$\Theta(n^{\log_3 3}) = \Theta(n^1)$

as $n^2 > n^1$ asymptotically

the $\Theta(T(n)) = \Theta(h(n))$

$\boxed{\Theta(n^2)}$

5. $T(n) = T(3n/4) + T(n/3) + n$

$T(n) \leq 2T(3n/4) + n$

$a = 2 \quad b = 4/3 \quad h(n) = n$

$O(n^{\log_{4/3} 2}) > n$

$\boxed{\Theta(n^{\log_{(4/3)} 2})}$

a)
N = either array's length

```
1. get median of A and B (am, bm)
2. if am = bm then median found
3. if am > bm
       A = A[0 : n/2]
       B = B[n/2 : n-1]
   else
       A = A[n/2 : n-1]
       B = B[0 : n/2]

4. repeat until n == 2

5. median =
(max(A[0], B[0]) + min(A[1], B[1]))/2
```

b)

```
1. get kth smallest elem in
   A and B (if k > m or n then sub with
                 m or n respectively)
   ak and bk (index not val)

2. d = ak + bk + 1
   if d = k return max(A[ak], B[bk])

   m = max(A[ak], B[bk])

3. if d = k+1 then m-1
   else m/2 ⇒ back to step 2
```

a) b = # of blocks = $\frac{n}{2} \times \frac{n}{2}$

lets say n = 64

32 × 32 blocks

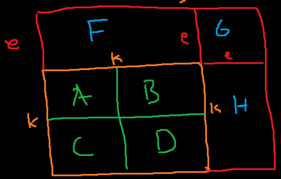
blit then recurse, or vice versa

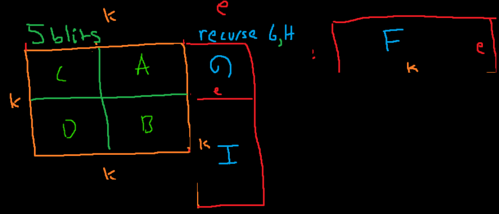
b) 5 blits when pow of 2

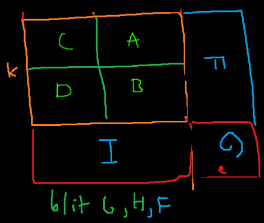
c) rather than seperating the pic into 4 block,

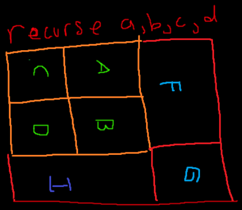
$n^2 = (k - ext)^2$ with

n being any whole num and k being any power of 2









d) k × k

3(e × k)

+ e × e

$k^2 + 3\boxed{ke} + e^2$

$O(n^2)$

e) O(n log n)

P3

```
B[0 → n-1]
1. L = 0  R = n-1

2. m = L+(r-L)/2

3. Lb = m-1   Rb = m+1
   if Lb < L or Rb > R → discard
   max = max(B[Lb], B[Rb])
4. if max < B[m]
       return B[m]
   else if B[rb] = max
       L = rb
   else
       R = Lb
5. go back to step 2
```

$\boxed{O(\log(n))}$

as this resembles binary search, and guaruntees to find a local max

P4) $x = 6$

1. Start at ends of first row & col

1	2	3	4 r
2	3	6	9
3	7	7	11
c 4	8	10	11

2. Check if $x \leq r$ or c

3. if not, iterate them to next row / col

1	2	3	4
2	3	6	9 c
3	7	7	11
4	r 8	10	11

go back 2 step

4. if so, start iterating both inwards until r or $c = x$

1	2	3	4
2	3	6	9 c
3	7	7	11
4	r 8	10	11

stop iterating after r==c as x is not in A

1	2	3	4
2	3	6 c	9
3	r 7	7	11
4	8	10	11

Since c==x x is in A!

$\boxed{P5}$

n = pos int > 1

create an empty map t

```
min chain(n)
    if n == 1 return 1
        A = list of all unique pairs
            that sum to n
        B = empty list
    for pair(i, j) in A
        if i in t, then it = t[i]
        else it = min chain(i)
        if j != i and j not in t
            then jt = min chain(j)

    if j == i then B.append(it)
        else B.append(it + jt)

    return min(B)
```

```python
#approximate min addition chain, lookin good! uses DP to run faster!
track = {}

def getPairs(n):
    A = []

    for i in range(1, n//2 + 1):
        A.append([i, n - i])

    return A

def minChain(n):
    global track

    if(n == 1):
        return 0

    A = getPairs(n)
    B = []

    for i in A:

        first = track.get(i[0])
        second = track.get(i[1])
        equals = i[0] == i[1]

        if first == None:
            first = minChain(i[0])
            track[i[0]] = first

        if second == None and not equals:
            second = minChain(i[1])
            track[i[1]] = second

        if equals:
            B.append(first + 1)
        else:
            B.append(first + second + 1)

    return min(B)

print(minChain(374))

print(track)
```