Problem #1

1. $k = 1023$

P_2 is going to select from #s $1 < i < 1023$.
Thus P_2 can make maximum of 1021 guess.
If P_2 guesses $j = i$ right away - game is over.

The optimal strategy for P_2 would be guessing using a methodic of a binary search - search through the given limit of #s by repeatedly ½ the search interval in half. If the value of the search key is less than the item in the middle of the interval, narrow the interval to the lower half. Otherwise, narrow it to the upper half. Repeat until the value is found. (Time complexity is $O(\log n)$).

Thus, if $1 < i < 1023$ we can guess the # of steps:
[for example, if $i = 123$]

1) $j = \lfloor 1023 : 2 \rfloor = 511 \rightarrow i < j$
2) $j = \lfloor 511 : 2 \rfloor = 255 \rightarrow i < j$
3) $j = \lfloor 255 : 2 \rfloor = 127 \rightarrow i < j$
4) $j = \lfloor 127 : 2 \rfloor = 63 \rightarrow i > j$
5) So we determined that $63 < i < 127$,
now we have $j = \lfloor ((127 - 63) : 2) \rfloor + 63 = 95 \rightarrow i > j$
6) Now we have $95 < i < 127 \Rightarrow j = \lfloor ((127 - 95) : 2) \rfloor + 95 = 111$
$\rightarrow i > j$
7) Now we have $111 < i < 127 \Rightarrow j = \lfloor ((127 - 111) : 2) \rfloor + 111 =$
$= 119 \rightarrow i > j$

8) Now we have $119 < i < 127 \Rightarrow$
$j = \lfloor((127-119):2)\rfloor + 119 = 123.$
At this step $j = i$.

Thus, we can assume that an optimal strategy for P_2 would be using a binary search as it takes approximately 8 steps in the average case scenario.

(2.) $K = 1023$ and P_1 tells P_2 before the first guess that i is even

- Thus, $\lfloor 1023:2 \rfloor = 511$ #s to guess from (it's the worst case scenario- 511 guesses)
- Again, if P_2 guesses $j = i$ right away- game is over. It took 1 guess.
  By using binary search, it approximately takes the same # of steps as 1st case - 1 or -2 steps, because every time dividing by 2 during binary search, we can shift to an even # if we happen to have a result as odd. So approximately, it would take 6-8 steps.

(3) $K = 1023$ and P_1 tells P_2 after answering the second guess that i is even

- In the worst case scenario it would take $1 + (\lfloor 1023:2 \rfloor) = 512$ guesses
- In the best case - 1 guess

- In the average case scenario, using binary search it would approximately take 6-8 steps again. (± 1# doesn't make much difference in binary search)

(4) $K = 2047$ and P_1 tells P_2 before the 1st guess that i is divisible by 8.

We have to calculate how many #s in 2047 are % by 8. $\Rightarrow \lfloor 2047 : 8 \rfloor = \underline{255}$

- In worst case, P_2 will make 255 guesses
- In the best case P_2 will make 1 guess ($j = i$)
- In the avg case, by using binary search, it would take approximately $\Rightarrow$

$\lfloor 255_1 : 2 \rfloor = \lfloor 127_2 : 2 \rfloor = \lfloor 63_3 : 2 \rfloor = \lfloor 31_4 : 2 \rfloor = \lfloor 15_5 : 2 \rfloor = \lfloor 7_6 : 2 \rfloor =$
$= \lfloor 3_7 : 2 \rfloor = 1$ Approximately 7 guesses

(5) $K = 255$ and P_1 tells P_2 before the 1st guess that i is a square #, $\sqrt{i} \in \mathbb{Z}^+$

Lets find all the square # between $1 < i < 255$

$\Rightarrow$ 4, 9, 16, 25, 36, 49, 64, 81, 100, 121, 144, 169, 196, 225.

There are 14 squares in $1 < i < 255$

- Thus, in the worst case scenario P_2 has to guess 14 times.
- In the best case alway 1 guess
- In the avg case, using binary search, P2 would make $14 : 2 = \lfloor 7_1 : 2 \rfloor = \lfloor 3_2 : 2 \rfloor = 1$ 3 guesses.

⑥ $k=64$ and P_1 tells P_2 before the 1st guess that i is a power of 2, that is, $i=2^p$ for some $p \geq 0$

Let's find all the # with power of 2 => $1 < i < 64$

$2^0=1$ (out of bound), $2^1=2$, $2^2=4$, $2^3=8$, $2^4=16$, $2^5=32$, $2^6=64$ (out of bound)

- So we have 5 guesses P_2 would have to make in the worst case
- 1 guess in the best case
- In the avg case, using binary search

$\Rightarrow \lfloor 5:2 \rfloor = 2 : 2 = 1$

P_2 would make 2 guesses.

Problem #2 We can modify merge sort so it computes the # of inversions in the input array in $\Theta(n \log n)$ by finding $L[i] > R[j]$, so that each element of $L[i \ldots n]$ would be an inversion with $R[j]$, since array L is sorted. Here's the pseudocode for this algorithm:

```
ToCountInversions (A, left, right)
1.  inversions = 0
2.  if left < right
3.      middle = ⌊(left + right) : 2⌋
4.      inversions = inversions + ToCountInversions(A, left, middle)
5.      inversions = inversions + ToCountInversions(A, middle+1, right)
6.      inversions = inversions + InsideInversions(A, left, middle, right)
7.  return inversions
```

```
InsideInversions(A, left, middle, right)
1.  n1 = middle - left + 1
2.  n2 = right - middle
3.  let L[1...n1+1] and R[1...n2+1] be new arrays
4.  for i = 1 to n1
5.      L[i] = A[left + i - 1]
6.  for i = 1 no n2
7.      R[i] = A[middle + i]
8.  L[n1+1] = R[n2+1] = ∞
9.  i = j = 1
10. inversions = 0
11. counted = false
12. for k = left to right
13.     if counted = false and L[i] > R[i]
14.         inversions = inversions + n1 - i + 1
15.         counted = true
16.     if L[i] ≤ R[j]
17.         A[k] = L[i]
18.         i = i + 1
19.     else A[k] = R[j]
20.         j = j + 1
21.         counted = false
22. return inversions
```

Problem #3

(1.) $T(1) = 5$ and $T(n) = T(n-1) + 7$ $\forall n > 1$

Solution:

$T(n) = T(n-1) + 7$

$T(n) = T(n-2) + 7 + 7$

$T(n) = T(n-3) + 7 + 7 + 7$

$T(n) = T(n-4) + 7 + 7 + 7 + 7$

$\Rightarrow T(n) = T(n-(n-1)) + (n-1)\cdot 7$

$T(n) = T(1) + (n-1)7$; $T(n) = 5 + 7n - 7$

$T(n) = 7n - 2 \Rightarrow T(n) = O(n)$

Proof by induction

Base case: $T(n) = 7n - 2$

$n = 1$, $T(n) = 7 \cdot 1 - 2 = 5$ true ✓

Assuming: for any $k > 1$, $T(n)$ is true
Show that $T(n+1)$ is also true

Induction: $T(n) = T(n-1) + 7$

$T(k+1) = T(k) + 7 = 5 + (k-1)(7) + 7 =$

$= 5 + 7((k-1)+1) = 5 + 7k$

Note: main solution to problem #3(1)

$T(k+1) = 5 + (k+1-1)7 = 5 + 7k$

proven

(2) $T(1) = 3$ and $T(n) = 2T(n-1)$

$T(2) = 2 \cdot T(1) = 6 = (2)(3)$
$T(3) = 2 \cdot T(2) = 12 = (2^2)(3)$
$T(4) = 2 \cdot T(3) = 24 = (2^3)(3)$

We can guess general formula $T(n) = (2)^{n-1} \cdot (3)$

Proof by induction

Base case: $T(n) = 2^{n-1} \cdot 3$

$T(1) = 2^{1-1} \cdot 3 = 3$ ✓ we have
$T(2) = 2^{2-1} \cdot 3 = 6$ ✓ a base

Assuming that for k our solution is true.
Show that it's also true for $T(k+1)$

Induction: $T(k) = (2^{k-1})(3)$

$T(n) = 2T(n-1)$, so $T(k+1) = 2T(k)$ (page #7)

$T(k+1) = 2 \cdot 2^{k-1} \cdot 3 = 2^k \cdot 3$ =

$T(n) = (2^{n-1})(3) = 2^{k+1-1} \cdot 3 = 2^k \cdot 3$

proven

problem #4

1. $T(n) = 8T\left(\frac{n}{2}\right) + n^3$

$a = 8, b = 2, f(n) = n^3$ = case 2.

$n^{\log_b a} = n^{\log_2 8} = n^3$

$f(n) \in \Theta(n^{\log_b a}) \Rightarrow T(n) \in \Theta(n^{\log_b a} \cdot \log n) =$

$= \Theta(n^3 \log n)$

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

$a = 2, b = 3, f(n) = n^2$ = case #3

$n^{\log_b a} = n^{\log_3 2} \approx n^{0.63} < n^2$

$a \cdot f\left(\frac{n}{b}\right) \leq c \cdot f(n)$ $\quad c < 1, n > n_0$

$2 \cdot \left(\frac{n^2}{3^2}\right) \leq c \cdot f(n)$

$\frac{2n^2}{9} \leq c f(n)$

$\frac{2n^2}{9} \leq \underset{c<1}{c} \cdot n^2$ is true for $c = \frac{2}{9}$

$\Rightarrow f(n) = \Omega(n^{\log_4(3)+\varepsilon})$, then

$T(n) = \Theta(f(n^2)) = \Theta(n^2)$

$T(n) = 7n - 2. \Rightarrow T(n) = O(n)$

Proof by induction:

Assume, $T(n) = O(n) \Rightarrow T(n) \leq cn$ for some const c.

Base case: when $n = 1$, $T(1) = 5$ & $c \cdot 1 = c$

Thus, we have $T(1) \leq c, \forall c \geq 5$

Thus, $T(n) = O(n)$ is true for $n = 1$.

Induction: Assuming $T(n)$ is true for $n-1$

$\Rightarrow T(n-1) = O(n-1) \Rightarrow T(n-1) \leq c(n-1)$

Now we will show that $T(n)$ is also true for n.

$T(n) = T(n-1) + 7$

$T(n) \leq c(n-1) + 7$

$T(n) \leq cn + (7 - c)$

let $c_1 n \geq cn + 7 - c$ for any const c_1 and c

Thus, we have $T(n) \leq c_1 n \Rightarrow \boxed{T(n) = O(n)}$ proved ✓

2nd solution to the problem #3 (1)

Can it be a valid solution?

Problem #5

Let Rec(1,n) be $T(n)$, then

$$T(n) = T\left(\frac{n+2}{3}\right) + T\left(\frac{n+5}{3}\right) + \log_2 n$$

Taking tight bound:

$$T(n) = T\left(\frac{n}{3}\right) + T\left(\frac{n}{3}\right) + \log n$$
$$= 2T\left(\frac{n}{3}\right) + \log n$$

Master Method: $a = 2, b = 3, f(n) = \log n$

$n^{\log_b a} = n^{\log_3 2} = n^{0.63} < \log n \Rightarrow$ case #3

$a \cdot f\left(\frac{n}{b}\right) \leq c f(n) \qquad c < 1, n > n_0$

$2 \cdot f\left(\frac{\log n}{3}\right) \leq c f(n)$

$\frac{2 \log n}{3} \leq c f(n)$

$\frac{2 \log n}{3} \leq c \log n$ is true for $c = \frac{2}{3}$

$\Rightarrow f(n) = \Omega(n^{\log_3 2 + \varepsilon})$, then $T(n) = \Theta(n^{\log_b a}) = \Theta(\log_3 2)$

Problem #6

For inputs where the length of the input integers is not power of 2 we can append 0's in the beginning, this is for the even length and if our inputs are of odd length, then we have to put floor $(\frac{n}{2})$ bits in the left half and

ceil $(\frac{n}{2})$ bits in the right half where n is the # of bits.

example: Let's say we have 2 input strings A & B of length not the power of 2, so the final value of A*B will become:

l - left, r - right

$$A \cdot B = 2^{2ceil(\frac{n}{2})} Al \cdot Bl + 2^{ceil(\frac{n}{2})} \cdot [(Al+Ar)(Bl+Br) - (Al \cdot Bl) - (Ar \cdot Br)] + Ar \cdot Br$$

where Al and Ar contain leftmost & rightmost $\frac{n}{2}$ bits of A and Bl and Br contain leftmost & rightmost $\frac{n}{2}$ bits of B.

Problem #7

To modify Quicksort to sort its input into nonincreasing order, we need to change the main condition A[j] <= X to A[j] >= X

```
Partition (A, p, r)
    X = A[r]
    i = p - 1
    for j = p to r - 1
        if A[i] >= X
            i = i + 1
            swap A[i] and A[j]
    swap A[i+1] and A[j]
    return i+1
```