TQ 1

① it computes an element of A, called s, such that the list made of elements of A with value $\leq s$ has length s.

② In the worst case the algorithm doesn't get the right s till the base case. The base case is the one that have as input $([\cdot], 1)$. In the worst scenario I remove 1 element from A every recursion tree among at the base case.
We have

$$T(n) = \underbrace{n}_{\text{set L}} + \underbrace{n}_{\text{set R}} + \underbrace{c}_{\text{the other operation}} + T(n-1) = 2n + c + T(n-1)$$

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

$$= \dots = 2\sum_{j=0}^{i} n-j + T(n-i) + c = (A).$$

I want to arrive a $T(1)$ which costs c_1.
So $i = n-1$

We have $2\sum_{j=0}^{n-1} n-j = 2\sum_{i=1}^{n} i = 2\frac{(n+1)n}{2} = n^2+n$

$$(A) = n^2 + n + \underbrace{c + c_1}_{k} = n^2 + n + k = O(n^2).$$

③ In the best scenario we catch the right s at the first time, so $T(n) = 2n + c = O(n)$.

TQ 2

① $T(n) = 1 + T(n/2) + T(n/2) + 3\left(\frac{n}{2}\right)$ ~ cost of swaplist

$= 2\,T(n/2) + 3\left(\frac{n}{2}\right) + 1$

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

$= 2^i\,T\left(n/2^i\right) + \frac{3}{2}\left(\sum_{j=0}^{i-1} \frac{n}{2^j}\right) + i = (*)$

by the hint $n = 2^k$ for a certain k

so i want to continue till $n/2^i = 1$.

(We have also by the hint $len(a) = 2^e$, with the implicit condition of $e \geq k$).

$(*) = 2^i\,T\left(2^k/2^i\right) + \frac{3}{2}\left(\sum_{j=0}^{i-1} \frac{2^k}{2^j}\right) + i = \dots$

$= 2^k\,T(1) + 3\left(\sum_{j=1}^{k} 2^{k-j}\right) + k =$

$= n + 3n\left(\sum_{j=1}^{k} \frac{1}{2^j}\right) + \log_2 n$ ~, because of $2^k = n$.

We remember that for $|x| < 1$ $\sum_{i=0}^{n} x^i = \frac{1 - x^{n+1}}{1 - x}$ *

* in our case

$\sum_{j=1}^{k} \frac{1}{2^j} = \sum_{j=0}^{k} \frac{1}{2^j} - 1 = \frac{1 - \left(\frac{1}{2}\right)^k \cdot \frac{1}{2}}{1/2} - 1 =$

$= 2 - \frac{1}{n} - 1 = 1 - \frac{1}{n}$

$T(n) = n + 3n\left(1 - \frac{1}{n}\right) + \log_2(n) = 4n + \log_2(n) - 3 = \mathcal{O}(n)$.

TQ 2 (2)

```
Function swapList(a, l, n):
    for i=0 to n/2:
        tmp = a[l+i]
        a[l+i] = a[l+n/2+i]
        a[l+n/2+i] = tmp
```

if we take a = [0,1,2,3,4,5,6,7], l=4, n=4.
We have:
for i=0

tmp = a[l+i] = a[4+0] = 4, a[4] = a[0] = 6, a[6] = tmp = 4

So after the first iteration, we have

[0,1,2,3,6,5,4,7].

for i=1 tmp = a[5] = 5, a[5] = a[7] = 7, a[7] = 5.
So after the second iteration we have.

[0,1,2,3,6,7,4,5]. and the function finishes its work.

So we have that swapList switch a[j] with a[j+n/2]
for j = l to l+n/2

We observe that n+l ≤ len(a).

let's see what happens when l=3 and n=4

[0,1,2,3,4,5,6,7]

i=0 a[3] = a[5] = 5, a[5] = 3; i=1 a[4] = a[6] = 6, a[6] = 4

↓

[0,1,2,5,6,3,4,7].

Let's see another situation: $\ell=2$ and $u=6$.

$[0,1,2,3,4,5,6,7]$

$i=0$ $a[2]=a[5]=5$, $a[5]=2$; $i=1$ $a[3]=6$, $a[6]=3$; $i=2$ $a[4]=a[7]=7$, $a[7]=4$

↓

$[0,1,5,6,7,2,3,4]$

Let's see what happens when we have $\ell=0$ and $u=8$.

$a[0]=a[4]=4$, $a[4]=0$; $a[1]=a[5]=5$, $a[5]=1$;

$a[2]=a[6]=6$, $a[6]=2$; $a[3]=a[7]=7$, $a[7]=3$

↓

$[4,5,6,7,0,1,2,3]$.

Function splitSwap(a, ℓ, u).
if $u \leq 1$: return
splitSwap(a, ℓ, u/2).
splitSwap(a, ℓ+u/2, u/2)
swapList(a, ℓ, u)

Let's take a look at the behaviour of this algorithm. Let's consider
$a=[0,1,2,3,4,5,6,7]$
$\ell=4$, $u=4$
Since the function call swapList we have the same condition:
$\ell+u \leq \text{len}(a)$.

splitswap(a,4,4)

splitswap(a,4,2) → splitswap(a,4,1) → swaplist(a,4,2) → a[4]=5, a[5]=4 → [a,1,2,3,5,4,6,7]

splitswap(a,6,2) → splitswap(a,6,1) → swaplist(a,6,2) → a[6]=7, a[7]=6. → [a,1,2,3,5,4,7,6]

swaplist(a,4,4)
a[4]=a[6]=7
a[6]=a[4]=5.
a[5]=a[7]=6.
a[7]=a[5]=4
→ [0,1,2,3,7,6,5,4]

So we have:

[0,1,2,3,4,5,6,7] —splitswap(a,4,4)→ [0,1,2,3,7,6,5,4].

So splitswap reverse the order of the sublist a[l:l+n], i.e a[l:l+n] = a[l+n-1:l-1:-1] if l≠0,

and

a[l:l+n] = a[l+n-1::-1] if l=0.

Let's see what if n=8. and len(a)=16

splitswap(a,4,8) → splitswap(a,4,4) → splitswap(a,4,2) → splitswap(a,4,1) → swaplist(a,4,2) → swaplist(a,4,4).

splitswap(a,4,8) → splitswap(a,8,4) → splitswap(a,8,2) → splitswap(a,8,1) → swaplist(a,8,2) → swaplist(a,8,4)

splitswap(a,4,8) → swaplist(a,4,8)

The algorithm can be optimized, in fact. considering $n = 2^k$ for a certain k, where we call splitswap(a,l,n) the function calls himself $2k$ times, and call sweplist $2(k-1)+1 = 2k+1$ times.

We can optimize the algorithm as it follows:

```
function splitswap(a,l,n):
    if n<1: return
    if l+n > len(a): return  # the condition we talk about to not exceed length with index
    if l==0:
        a[l:l+n] = a[l+n-1::-1]
    else:
        a[l:l+n] = a[l+n+1:l-1:-1]
```

TQ3 (a) Fix $n=3$ and $W=5$. We have that $e_i = (v_i, w_i)$

Let's say $e_1=(4,2)$, $e_2=(5,3)$, $e_3=(10,5)$.

Following the heuristic we obtain the solution $\{e_1, e_2\}$ but it not maximizes the values. because $v_1+v_2 < v_3$.

(b) $n=3$, $W=10$

$(e_1, e_2, e_3) = ((10,9), (6,5), (5,5))$.

Following the heuristic the solution is $\{e_1\}$ but it not maximizes the values. the optimal solution is $\{e_2, e_3\}$ since

$v_1 = 10 < 6+5 = v_2+v_3$.

$n=3,\ W=10$

(C) I choose an element $(v_i, w_i) = e_i$, which will be one of the three elements, with a certain rapport $\frac{v_i}{w_i}$. Let's say that $\frac{v_i}{w_i} = \frac{8}{10} = \frac{4}{5}$. We can obtain this rapport with $(8,10)$ and also with $(4,5)$.
We take $(4,5)$ so that we will be able to overcome the values. So $e_1 = (4,5)$.
Let's take e_2 with a ~~[illegible]~~ slightly little rapport, let's say $\frac{7}{10}$. 7 and 10 are coprime so $e_2 = (7,10)$.
Let's observe that e_2 covers all the budget as e_3 we can take a couple of v and w. Such that the rapport is $< 7/10$ and $v_1 + v_3 < v_2$. So we choose $e_3 = (7,5)$.
the solution provided by heuristic is $\{e_1, e_3\}$ but the optimal is $\{e_2\}$.