Sorting

(1) Prove that searching for the minimum among n elements requires $n-1$ comparisons. Hint: For any algorithm and at any time, the set of n elements can be partitionned into:

- $A = \{$elements candidate for being the minimum$\}$
- $B = \{$elements which cannot be candidate for being the minimum anymore$\}$

When the algorithm starts, all of its elements belong to A and when it stops, A contains a single element.

(2) Bubble sort improvement: The best case time bound can be improved by testing if there weren't any swap. In this case, we can exit the loops and end the method. Provide an algorithm for implementing this improvement and evaluate the best-time complexity.

(3) Propose an efficient algorithm ($3n/2 + o(1)$ comparisons) for finding simultaneously the maximum and the minimum among n elements (for n even).

(4) We consider the following algorithm for sorting n elements:

```
def asort!
 def lasort(left,right)
   n=(right-left+1)
   case
     when n==1 return self
     when n==2 self.swap!(left,right) if self[left]>self[right]
      else k=n.div(3)
       self.lasort(left,right-k)      #sorts the first 2/3 of the array
       self.lasort(left+k,right)      #sorts the last 2/3 of the array
       self.lasort(left,right-k)      #sorts the first 2/3 of the array
     end
 end
 self.lasort(0,self.lenght-1)
 self
end
```

Prove that `asort!` effectively sorts an array; give a recurrence relation on its running time in the worst case and solve it.