1.

$a)$

$For\ j \leftarrow 1\ to\ n - 1:\ \#\ main\ loop\ that\ iterates\ n - 1\ times:$

$\quad For\ i \leftarrow n\ to\ 2:\ \#\ visits\ the\ elements\ of\ A\ in\ decreasing\ order\ of\ indices:$

$\quad\quad If\ A[i] > A[i - 1]\ then\ \#\ compares\ A[i]\ with\ A[i - 1]:$

$\quad\quad\quad swap(A[i], A[i - 1])\ \#\ if\ the\ statement\ is\ true,\ do\ the\ swap$

$b)$

$In\ such\ algorithm\ it\ guaranteed\ that\ A[i - 1]\ is\ larger\ than\ A[i]\ after\ first\ i\ passed$

$for\ base\ case:\ n = 2$

$if\ A[2] > A[1]\ then\ do\ the\ swap\ otherwise\ the\ array\ already\ sorted\ so\ it\ is\ correct$

$Assuming\ that\ algorithm\ sorts\ the\ array\ size\ of\ n - 1 >= 2\ correctly,\ such\ that[A[2]\ to\ A[n]]\ is$
$in\ descending\ order$

$first\ add\ a\ largest\ number\ in\ A,\ index\ as\ 1,\ with\ above\ induction,\ this\ number\ A[1]\ would$
$be\ larger\ than\ A[2]\ as\ guaranteed\ that\ A[i - 1]\ is\ larger\ than\ A[i]\ after\ first\ i\ passed$
$As\ now\ [A[2]\ to\ A[n]]\ is\ sorted\ in\ descending\ order(assumed)$
$[A[1]\ to\ A[n]]\ would\ also\ be\ sorted\ in\ descending\ order$

$if\ the\ adding\ number\ \alpha\ is\ not > A[2]\ and\ still\ adding\ in\ index\ 1\ as\ A[1].\ and\ for\ a\ number\ k,\ k \leq n$
$such\ that\ A[k - 1] > \alpha \geq A[k],\ as\ guaranteed\ above,\ after\ k - 1\ times\ iterations$
$each\ time\ of\ A[i = k\ to\ 2] > \alpha$
$such\ that\ \alpha\ would\ move\ to\ A[k - 1],\ with\ such\ induction\ A[1]...A[n]would\ be\ sorted\ in\ decreasing\ order$
$so\ the\ algorithm\ is\ correct\ as\ n - 1\ case\ correct\ implied\ n\ case\ correct$

$c)$

$As\ this\ algorithm\ would\ run\ n - 1\ times\ iterations\ and\ in\ each\ iterations\ it\ will\ do\ n - 1\ times$
$comparisons,\ so\ in\ total\ the\ worst\ case\ running\ time\ would\ be\ (n - 1)(n - 1) = n^2 - 2n + 1$
$so\ the\ bound\ on\ the\ worst - case\ running\ time\ would\ be\ O(n^2)$

2.

a)

```
let B be a list for storing the answer
function permutation(temporary array α):
    if length of α equals to length of A then
        add α in B
        return # terminate the function call
    For i ← 1 to n: # iterate n times and add element i to temporary array α
        if A[i] is not in α then
            A[i] append in α
            permutation(α)# recursively call the function with added element
permutation(empty array [])#initial call of the function
the return would be B as a array storing all the permutation array
```

b)

Recurrence for the running time of recursive calling the function in(a)

$T(n) = nT(n-1)$ *with boundary condition* $T(1) = 1$

by solving the recurrence $T(n-1) = (n-1)T(n-2)$

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

In general case $T(n) = n!$ *such that* $T(n) = n! = O(n!) = \Omega(n!)$ *such that* $T(n) = \Theta(n!)$

3.

$a)\quad \frac{n^2}{log(n)} = O(n^2)$

$b)\ 2^{log(n)} = O(n^3)$

$c)\ let\ k = log(n),\ log(k) = O(k),\ so\ log(log(n)) = O(log(n))$

$d)\ as\ 4^{log_2 n} = n^{log_2 4} = n^2\ such\ that\ n^2 = \Theta(n^2)\ so\ n^2 = \Theta(4^{log_2 n}),\ 4^{log_2 n} = \Theta(n^2)$

$e)\ (logn)^2 = O(n^{1/3})$

4. Master theorem

$T(n) = aT\left(\frac{n}{b}\right) + n^d$ if $d < \log_b a$, $\Theta(n^{\log_b a})$

$T(n) = n^{\log_b a} + n^d \left(\frac{\left(\frac{a}{b^d}\right)^{\log_b n} - 1}{\left(\frac{a}{b^d}\right) - 1} \right)$ if $d = \log_b a$, $\Theta(n^d \log n)$

if $d > \log_b a$, $\Theta(n^d)$

a) $a = 4$ $b = 2$ $d = 1$

$1 < \log_2 4$ so $d < \log_b a$

So the upper bound be $O(n^{\log_2 4}) = O(n^2)$

---

b) $a = 4$ $b = 2$ $d = 3$

$3 > \log_2 4$ so $d > \log_b a$

So the upper bound would be $O(n^3)$

c) $a = 1 \quad b = 3 \quad d = \frac{1}{2}$

$\frac{1}{2} > \log_3 1$ so $d > \log_b a$

So the upper bound be $O(n^{\frac{1}{2}})$

---

d) $T(n) \quad 5T(\frac{n}{4}) + n\log n$

$T(\frac{n}{4}) = 5T(\frac{n}{4^2}) + \log(\frac{n}{4})(\frac{n}{4})$

$T(n) = 5(5(T(\frac{n}{4^2}) + (\frac{n}{4})\log(\frac{n}{4})) + n\log n$

$= 5^2(T(\frac{n}{4^2}) + (\frac{n}{4})\log(\frac{n}{4})) + n\log n$

$\vdots \quad = 5^2 T(\frac{n}{4^2}) + 5(\frac{n}{4})\log(\frac{n}{4}) + n\log n$

In general case. as $4^i = n$

$i = \log_4 n$

$T(n) = 5^i T(\frac{n}{4^i}) + n \sum_{i=0}^{\log_4 n - 1} (\frac{5}{4})^i \log(\frac{n}{4^i})$

as i goes to $\log_4 n$

$T(n) = n^{\log_4 5} + \underbrace{\sum_{i=0}^{\log_4 n - 1} (\frac{5}{4})^i}_{\text{constant}} \sum_{i=0}^{\log_4 n - 1} \log(\frac{n}{4^i}) n$

$\downarrow$

$\log(n \sum_{i=0}^{\log_4 n - 1} (\frac{1}{4})^i) n$

$\frac{1 - (\frac{1}{4})^{\log_4 n}}{1 - \frac{1}{4}} = \frac{4 - \frac{4}{n}}{3}$

$\log(\frac{4n-4}{3}) n$

simplify to

$\Rightarrow T(n) = n^{\log_4 5} + (\text{constant})(n)(\log n)$

as $n^{\log_4 5}$ grow faster than $n\log n$

so $T(n) = O(n^{\log_4 5})$

e)

$T(n^{\frac{1}{2}}) = T(\lfloor n^{\frac{1}{2^2}} \rfloor) + 1$

$T(n) = (T(\lfloor n^{\frac{1}{2^2}} \rfloor) + 1) + 1$

For general case it inducted to be

$T(n) = (T(\lfloor n^{\frac{1}{2^i}} \rfloor) + i$ as i to be the minimal of levels to base case $T(1)$

as $\lfloor \ \rfloor$ is a floor function while

$1 \le n^{\frac{1}{2^i}} < 2$ it would be $= 1$

So $n^{\frac{1}{2^i}} < 2 \rightarrow \frac{1}{2^i} < \frac{\log 2}{\log n}$

$\rightarrow \log_2 n < 2^i \quad \log_2 \log_2 n < i \quad i = \lceil \log_2 n \log_2 n \rceil$

as inducted, $\log_2 n \log_2 n$ be the minimal levels to base $T(1)$ $T(n) = 1 + \log_2 \log_2 n$

Such that $T(n) = O(\log \log n)$

5.

use a divide and conquer algorithm

#binary search

$l \leftarrow 1$

$r \leftarrow n$

$While\ l\ <\ r$:

$m\ =\ (l\ +\ r)//2$ *#divide into* 2 *subarray and floor it if* $(l\ +\ r)$*is not divisible by* 2

$If\ r\ is\ l\ +\ 1$:

$return\ A[r]$

$if\ A[m]\ >\ A[m\ +\ 1]$: *#check if the unique minimum stay in the right sides or left*

#if $A[m]\ >\ A[m\ +\ 1]$ *it means the unique minimum would stay in right side*

$l\ =\ m$ *#then subtract into a half size array in this case as right side*

$else$:

$r\ =\ m$ *#otherwise left side*

For $n\ =\ 3$, *is a base case , by* $[n_1, n_2, n_3]$ *such that by definition* n_2 *appear to be the unique minimum*

then $A[m]\ = n_2$, $m\ =\ 2$, $A[m]\ <\ A[m\ +\ 1]$ *so* $r\ =\ m$, $A[r]\ = n_2$, $A[l]$ *still be* n_1 *and* $l\ =\ 1$

so $r\ =\ l\ +\ 1$ *return* $A[r]\ = n_2$ *to be the unique minimum*

For $n\ >\ 3$

Assuming the algorithm can correctly find the unique minimal γ *of index* r_γ *in a n size array A*

by appending a new element α *, there will be* 2 *cases*

1. $\alpha\ >\ \gamma$

By the definition of convex function, α*would live in* $A[1]$*to* $A[r_\gamma\ -\ 1]$ *or* $A[r_\gamma\ +\ 1]$*to* $A[n]$

with the algorithm running, either the α *would get eliminated in the progress of division to subarray or if* γ *next to* α*, take m as* r_γ *,* $A[r_\gamma]\ <\ A[r_\gamma\ +\ 1]$*so* $r\ = r_\gamma$ *as* $l\ = r_\alpha$ *,* $r_\gamma = r_\alpha\ +\ 1$*, return* $A[r_\gamma]\ =\ \gamma$

such that γ *would still be the unique minimum*

2. $\alpha\ <\ \gamma$

By the definition of convex function, α *would live in next to* γ*, take* $m\ = r_\gamma$ *then* $A[r_\gamma]\ >\ A[r_\gamma\ +\ 1]\ =\ A[r_\alpha]$

as now $l\ = r_\gamma$ *with* $r_\alpha = r_\gamma\ +\ 1$*, the algorithm would return* $A[r_\alpha]$*which to be unique minimum* α

so in two cases inducted that the algorithm is correct as true in case n implies case $n\ +\ 1$ *is true*

As in each loop do one comparison of $A[m]$ *and* $A[m\ +\ 1]$ *then divided in* $A[1...\ m]$ *and* $A[m\ +\ 1...\ n]$

so $T(n)\ =\ T(n/2)\ +\ 1$ *by master theorem* $a\ =\ 1,\ b\ =\ 2,\ d\ =\ 0$

$0\ =\ log_2 1$

$T(n)\ =\ O(log(n))$

6.

The algorithm would base on divide every polynomial function into a quadratic function and a constant term such that we can use the property $(ax + c)(bx + d) = (ab)x^2 + (ad + bc)x + (cd)$
with further induction $(ad + cb) = (a + c)(b + d) - (ab) - (cd)$
in such case we only need to compute$(ab), (bc)$ *and*$(a + c)(b + d)$ *to obtain the coefficient of* x^2, x^1 *and* x^0

For $p(x)$ *and* $q(x)$ *are polynomial function as it can divide to*

$p = (p_0)x + (p_1)x^0$

$q = (q_0)x + (q_1)x^0$

$pq = (p_0q_0)x^2 + (p_0q_1 + q_0p_1)x + (p_1q_1)x^0$

as inducted above

$(p_0q_1 + q_0p_1) = (p_0 + p_1)(q_0 + q_1) - (p_0q_0) - (p_1q_1)$

$pq = (p_0q_0)x^2 + ((p_0 + p_1)(q_0 + q_1) - (p_0q_0) - (p_1q_1))x + (p_1q_1)x^0$

A simple version code would be like
function $A(p, q, n)$:

$P_0Q_0 = A((p[n]x^{n-1} + p[n - 1]x^{n-2} + p[n - 2]x^{n-3} +...), (q[n]x^{n-1} + q[n - 1]x^{n-2} + q[n - 2]x^{n-3} +...))$

$P_1Q_1 = A((p[0]), (q[0]))$

$(P_0 + P_1)(Q_0 + Q_1)$

$= A(p[n]x^{n-1} + p[n - 1]x^{n-2} + p[n - 2]x^{n-3} +...+ p[0]), (q[n]x^{n-1} + q[n - 1]x^{n-2} + q[n - 2]x^{n-3} +...+ q[0]))$

return $(P_0Q_0)x^2 + ((P_0 + P_1)(Q_0 + Q_1) - (P_0Q_0) - (P_1Q_1))x + (P_1Q_1)x^0$

by recursively calling the function with input $p_0q_{0,}(p_0 + p_1)(q_0 + q_1)$*and* (p_1q_1)

$by\ merging\ back\ and\ return\ \ with\ above\ equation,\ we\ can\ get\ all\ the\ coefficient\ up\ to\ 2n\ terms$

$By\ assuming\ n\ is\ a\ power\ of\ 2,\ n\ =\ 2^k$
$there\ are\ 3\ function\ calling\ divisions\ of\ each\ size\ of\ 2\ also\ merging\ with\ above\ size\ n$
$so\ T(n)\ =\ 3T(n/2)\ +\ O(n)$
$by\ master\ theorem,\ a\ =\ 3, b\ =\ 2, d\ =\ 1,\ T(n)\ =\ \Theta(n^{log_b a}) if\ d\ <\ log_b a$
$1\ <\ log_2 3,\ so\ T(n)\ =\ \Theta(n^{log_2 3})\ =\ O(n^{log_2 3})\ in\ such\ case\ less\ than\ \Theta(n^2) and\ meet\ o(n^2)$

$For\ n\ =\ 0, is\ the\ base\ case\ returning\ P[0]Q[0]$
$For\ n\ =\ 1,\ also\ true\ with\ returning\ (P[0]Q[0])x^2\ +\ (P[0]Q[1]\ +\ P[1]Q[0])x\ +\ (P[1]Q[1])x^0$
$For\ n\ >\ 1\ assuming\ the\ algorithm\ would\ return\ correct\ answer\ with\ recursively\ divide\ the\ polynomial\ into$
$P[n]x^n\ +\ p[n\ -\ 1]x^{n-1}\ +\ p[n\ -\ 2]x^{n-2}\ +....+\ P[0]x^0$
$(p[n]x^{n-1}\ +\ p[n\ -\ 1]x^{n-2}\ +\ p[n\ -\ 2]x^{n-3}\ +...)x\ +\ p[0]x^0$
$P_0 =\ (p[n]x^{n-1}\ +\ p[n\ -\ 1]x^{n-2}\ +\ p[n\ -\ 2]x^{n-3}\ +...)x,\ P_1 =\ p[0]x^0$
$as\ doing\ the\ same\ thing\ to\ Q$
$we\ can\ get\ P_0 Q_0 =\ (p[n]x^{n-1}\ +\ p[n\ -\ 1]x^{n-2}\ +\ p[n\ -\ 2]x^{n-3}\ +...)(q[n]x^{n-1}\ +\ q[n\ -\ 1]x^{n-2}\ +\ q[n\ -\ 2]x^{n-3}\ +...)$
$P_1 Q_1 =\ (p[0])(q[0])$
$(P_0 + P_1)(Q_0 + Q_1)$
$=\ (p[n]x^{n-1}\ +\ p[n\ -\ 1]x^{n-2}\ +\ p[n\ -\ 2]x^{n-3}\ +...+\ p[0])(q[n]x^{n-1}\ +\ q[n\ -\ 1]x^{n-2}\ +\ q[n\ -\ 2]x^{n-3}\ +...+\ q[0])$
$with\ above\ assumption\ is\ correct\ we\ can\ see\ for\ n\ +\ 1\ case$
$P_0 Q_0 =\ (p[n\ +\ 1]x^n\ +\ p[n]x^{n-1}\ +\ p[n\ -\ 1]x^{n-2}\ +...)(q[n\ +\ 1]x^n\ +\ q[n]x^{n-1}\ +\ q[n\ -\ 1]x^{n-2}\ +...)$
$as\ let\ k\ =\ n\ +\ 1$
$we\ get\ (p[k]x^{k-1}\ +\ p[k\ -\ 1]x^{k-2}\ +\ p[k\ -\ 2]x^{k-3}\ +...)(q[k]x^{k-1}\ +\ q[k\ -\ 1]x^{k-2}\ +\ q[k]x^{k-3}\ +...)$
$(P_0 + P_1)(Q_0 + Q_1)\ =$
$(p[k]x^{k-1}\ +\ p[k\ -\ 1]x^{k-2}\ +\ p[k\ -\ 2]x^{k-3}\ +...+\ p[0])(q[k]x^{k-1}\ +\ q[k\ -\ 1]x^{k-2}\ +\ q[k\ -\ 2]x^{k-3}\ +...+\ q[0])$
$same\ as\ the\ n\ term\ so\ it\ is\ correct$
$P_1 Q_1\ still\ be\ (p[0])(q[0])$
$such\ that\ if\ n\ case\ is\ correct\ indicate\ n\ +\ 1case\ is\ also\ correct\ so\ the\ algorithm\ works\ correctly$