Part A. 1.

$f_1(n) = \sqrt{2^{\sqrt{n}}} = (2^{\sqrt{n}})^{\frac{1}{2}}$ $f_2(n) = (\log n)^{\pi}$ $f_3(n) = n^{(3\log n)^2}$ $f_4(n) = \pi^n$

$f_5(n) = \sum_{i=1}^{i=n} c\,i^2$ where c is a positive constant.

Since logarithms are increasing and do not change the increasing rate (growth), applying the logarithms to functions.

1) $\log f_1(n) = \log[(2^{\sqrt{n}})^{\frac{1}{2}}] = \frac{1}{2}\log(2^{\sqrt{n}}) = \frac{\sqrt{n}}{2}$ (since $\log x$ is on the base of 2).

$= \Theta(\sqrt{n})$

2). $\log(f_2(n)) = \log[(\log n)^{\pi}] = \pi \log(\log n) = \Theta(\log(\log n))$

3). $\log(f_3(n)) = \log[n^{(3\log n)^2}] = (3\log n)^2 \log n = 2^3 (\log n)^3 = \Theta((\log n)^3)$

4). $\log(f_4(n)) = \log(\pi^n) = n \cdot \log\pi = \Theta(n)$

5). $\log(f_5(n)) = \log(\sum_{i=1}^{i=n} c\,i^2) = \log(c \cdot \frac{n(n+1)\cdot(2n+1)}{6}) = \Theta(\log n^3) = \Theta(3\log n)$

It's obvious that $\sqrt{n} < n$ (① < ④) Based on given table in slide.

Since $n > \log n$ and logarithm is increasing, $\log n > \log(\log n)$ (⑤ > ②)

$(\log n)^3 > \log n$. Let $\log n = x$, $x^3 > x$ for $x > 1$. (③ > ⑤)

Until now, we get ① < ④, ③ > ⑤ > ②

We can compare function 1 and function 3:

By taking logarithm: $\log(\sqrt{n}) = \frac{1}{2}\log n$ Since $\log n > \log(\log n)$ (previous proof).

$\log(\log n)^3 = 3\log(\log n)$ ① > ③.

$\Rightarrow$ ② < ⑤ < ③ < ① < ④.

So the answer is $f_2(n) < f_5(n) < f_3(n) < f_1(n) < f_4(n)$.

---

2. a). $T(n) = T(\lfloor \frac{n}{3} \rfloor) + T(\lfloor \frac{n}{6} \rfloor) + n\sqrt{\log n}$

Since $T(\lfloor \frac{n}{3} \rfloor) \geq T(\lfloor \frac{n}{6} \rfloor)$, $2T(\lfloor \frac{n}{6} \rfloor) + n^{\sqrt{\log n}} \leq T(n) \leq 2T(\lfloor \frac{n}{3} \rfloor) + n^{\sqrt{\log n}}$

①. For left side, $2T(\lfloor \frac{n}{6} \rfloor) + n^{\sqrt{\log n}} \leq T(n)$.

Using the Master theorem: $a=2$ $b=6$ $f(n) = n^{\sqrt{\log n}} = 2^{\log_2(n^{\sqrt{\log n}})} = 2^{(\log n)^{\frac{3}{2}}}$

$g(n) = n^{\log_b a + \varepsilon} = n^{\log_6 2 + \varepsilon}$ $\lim_{n\to\infty} \frac{f(n)}{g(n)} = \lim_{n\to\infty} \frac{n^{\sqrt{\log n}}}{n^{\log_6 2 + \varepsilon}} = \infty$

$\Rightarrow$ ~~$g(n) = \Omega(f(n))$~~ $f(n) = \Omega(g(n))$

If we want $a \cdot f(\frac{n}{6}) \le c \cdot f(n)$ for some constant $c<1$ and $n>n_0$.

let $c=\frac{1}{2}$, $a f(\frac{n}{6}) = 2 \cdot 2^{(\log \frac{n}{6})^{\frac{3}{2}}} \le c \cdot f(n) = \frac{1}{2} \cdot 2^{(\log n)^{\frac{3}{2}}} = 2^{(\log n)^{\frac{3}{2}}-1}$

$\Rightarrow (\log \frac{n}{6})^{\frac{3}{2}} + 1 \le (\log n)^{\frac{3}{2}} - 1$

$2 \le (\log n)^{\frac{3}{2}} - (\log \frac{n}{6})^{\frac{3}{2}}$

Since $(\log n)^{\frac{3}{2}}$ has larger growth compared to $(\log \frac{n}{6})^{\frac{3}{2}}$, when n larger.

$(\log n)^{\frac{3}{2}} - (\log(\frac{n}{6}))^{\frac{3}{2}}$ increases.

Select $n=100$, it fits, and also fits for $n>100$.

$\Rightarrow$ LHS $= \theta(n^{\sqrt{\log n}})$ By master theorem rule 3.

(2). For right side, $T(n) \le 2T(\lfloor \frac{n}{3} \rfloor) + n^{\sqrt{\log n}}$

Using the Master theorem: $a=2$ $b=3$ $f(n) = n^{\sqrt{\log n}} = 2^{(\log n)^{\frac{3}{2}}}$

$g(n) = n^{\log_b a + \varepsilon} = n^{\log_3 2 + \varepsilon}$ $\lim_{n\to\infty} \frac{f(n)}{g(n)} = \lim_{n\to\infty} \frac{n^{\sqrt{\log n}}}{n^{\log_3 2+\varepsilon}} = \infty$

$\Rightarrow f(n) = \Omega(g(n))$.

If we want $a \cdot f(\frac{n}{3}) \le c \cdot f(n)$ for some constant $c<1$ and $n>n_0$.

let $c=\frac{1}{2}$. $a \cdot f(\frac{n}{3}) = 2^{(\log \frac{n}{3})^{\frac{3}{2}}+1} \le c \cdot f(n) = \frac{1}{2} \cdot 2^{(\log n)^{\frac{3}{2}}} = 2^{(\log n)^{\frac{3}{2}}-1}$

$\Rightarrow (\log \frac{n}{3})^{\frac{3}{2}} + 1 \le (\log n)^{\frac{3}{2}} - 1$

$2 \le (\log n)^{\frac{3}{2}} - (\log \frac{n}{3})^{\frac{3}{2}}$

Since $(\log n)^{\frac{3}{2}}$ has larger growth compared to $(\log \frac{n}{3})^{\frac{3}{2}}$, when n larger

$(\log n)^{\frac{3}{2}} - (\log \frac{n}{6})^{\frac{3}{2}}$ increases.

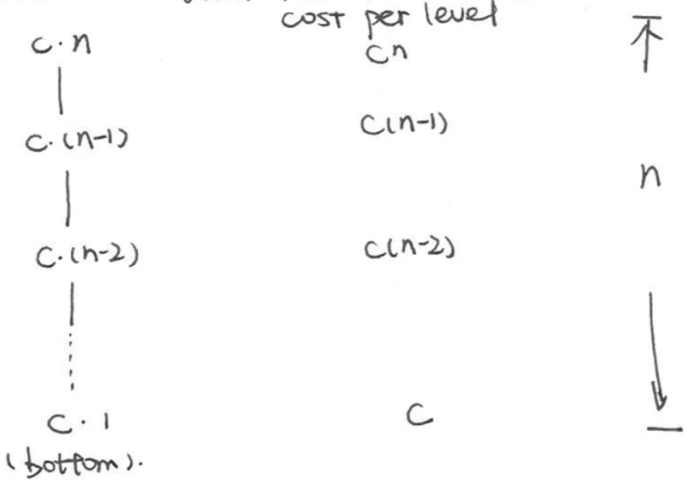
Select $n=5$, it fits and also fits for $n>5$.

$\Rightarrow$ RHS $= \theta(n^{\sqrt{\log n}})$ By master theorem rule 3.

$\Rightarrow T(n) = \theta(n^{\sqrt{\log n}}) = O(n^{\sqrt{\log n}})$.

---

b). $T(n) = T(n-1) + n$ with $T(1) = 1$. Using the recurrence tree.



The total cost $= \sum_{i=1}^{n} c \cdot i = c \cdot \sum_{i=1}^{n} i = c \cdot \frac{(n+1) \cdot n}{2} = O(n^2)$

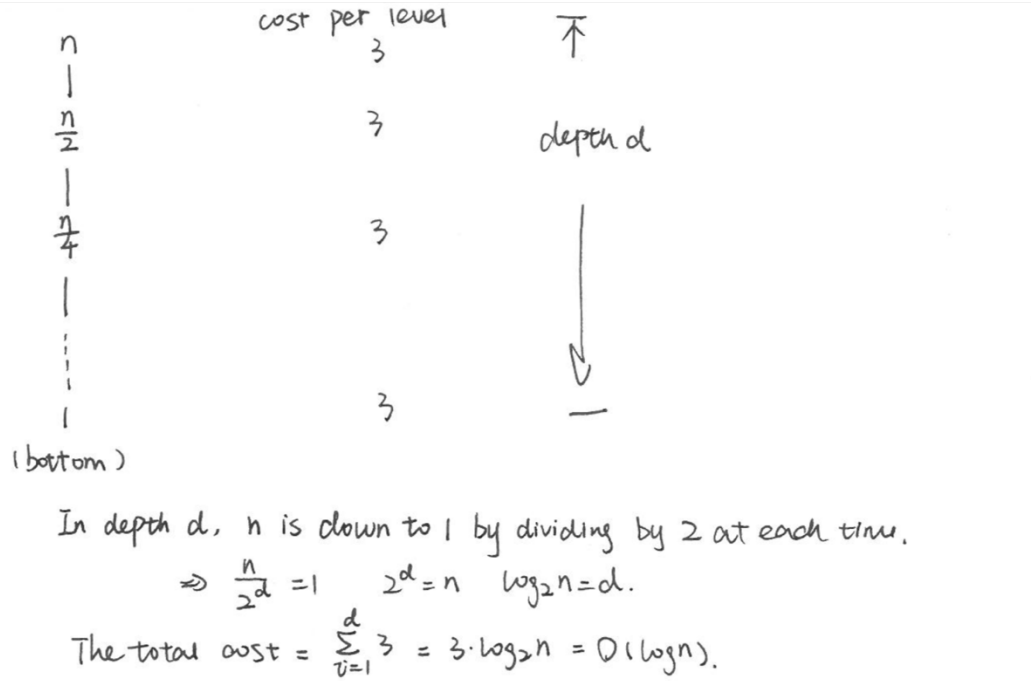
So the asymptotic upper bound is $O(n^2)$

3. a). We can use the binary search algorithm to let the worst case of searching is O(log n).
Following is the pseudocode:

```
binary_search (A, n, k):
        L = 0;
        R = n - 1;
        while L != R:
                m := ceiling(L + R) / 2;
                if A[m] > k:
                        R = m -1;
                else:
                        L = m;
        if A[L] == k:
                return L;
        else:
                return null;
```

1. A is the sorted array with length n. k is the value to be searched for index.
2. L is the left bound of the search and R is the right bound of the search. The initial search is the whole array, so L=0, R=n-1.
3. Only when L is not equal to R, the while loop runs. If the middle value A[m] is larger than the search value, only the left side (by setting the R to m-1) of current array will be searched. If the middle value A[m] is less than or equal to the search value, only the right side (by setting the L to m) of the current array will be searched.
4. Once while loop finishes, we check whether the only left value that L and R points to is the search value. If it is, the L and R is the index of the search value, return it. If not, it means the search value is not in the array and return null.

b). The algorithm used in part a can be written to $T(n) = T(n/2) + 3$ with $T(1) = 4$. Using the recursion to calculate the total cost.



Part B:

1. The designed system is used to find the best position of the light rail network which minimizes the total length of highways that connects each AIT's suburb to the rail network. The distance between a suburb and the light rail is the horizontal distance from the suburb centre to the light rail.
   Input: A set of suburb centres' coordinates (x_i, y_i)
   The size of area: L, W
   The number of total suburbs n
   Output: The best position of the light rail y (y-axis value)
   Criteria: $y_i \in L$, the total distance = min $(\sum_{i=1}^{i=n}|(y - y_i)|)$
   The best solution is the median value of all suburb centres' y value. Consider the following cases:
   1. If there is only one suburb existing, the minimum distance is 0 (by setting the light rail at the suburb centre).
   2. If there are two suburbs existing, setting the light rail at any place between two suburbs return the minimal total distances.
   3. If there are more than two suburbs existing and the number is even, we can pair the suburbs and set the light rail between each pair of suburbs. To achieve so, the median value satisfies.
   4. If there are more than three suburbs existing and the number is odd, we can pair suburbs and left a single suburb. If we can set the light rail at single suburb and this suburb must between all pairs of suburbs. The minimal distance achieves which is the median of the array.

2. A).
   merge (A, l, m, r)
   1. n1 = m – l + 1
   2. n2 = r – m
   3. let L[1..n1+1] and R[1..n2+1] be new arrays

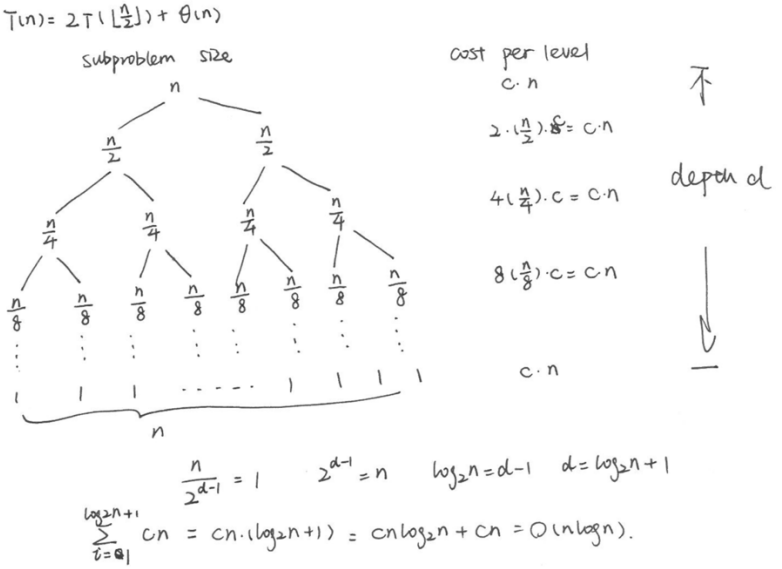
```
4.  for i = 0 to n1
5.      L[i] = A[l+i]
6.  for j = 0 to n2
7.      R[j] = A[m+1+j]
8.  i = 0
9.  j = 0
10. k = 1
11. while i < n1 and j < n2:
12.     if L[i] <= R[j]
13.         A[k] = L[i]
14.         i = I + 1
15.     else A[k] = R[j]
16.         j = j + 1
17. while i < n1:
18.     A[k] = L[i]
19.     i = i + 1
20.     k = k + 1
21. while j < n2:
22.     A[k] = R[j]
23.     j = j + 1
24.     k = k + 1
```

mergeSort (A, l, r)

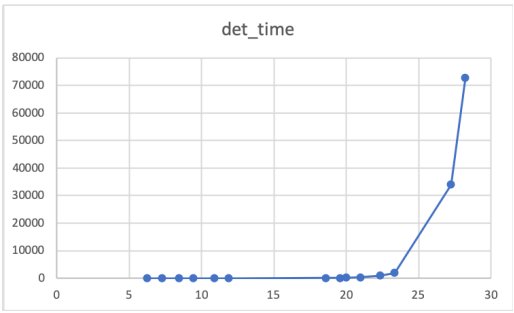
```
1.  if l < r
2.      m = l + (r-l)/2
3.      mergeSort (A, l, m)
4.      mergeSort (A, m+1, r)
5.      merge( A, l, m, r)
```

B). The above pseudo code can be written as:

$$T(n) = \begin{cases} \theta(1) & if\ n = 1 \\ 2T(\lfloor \frac{n}{2} \rfloor) + \theta(n) & if\ n > 1 \end{cases}$$



D). Seven more tests are generated for testing. For Deterministic_actual_graph', the x-axis is logn where n is the size of array, y is the program running time.



Deterministic_actual_graph

Since the x-axis is logt, the graph is compared with $x \cdot e^x$. The actual graph is very similar to the expected graph which shows the code's time complexity is O(nlogn).

3. A). partition (A, l, r, p)
    1. pivot = A[p]
    2. swap (A[p], A[r])
    3. p' = l
    4. For i = l to r
    5. If A[i] <= pivot
    6. Swap A[i] with A[p']
    7. p' = p' + 1
    8. Swap A[p'] with A[r]
    9. return p'

quickselect(A, l, r, k)

1. If l == r
2. return A[l]
3. pivot = random(l, r)
4. pivot_Index = partition(l, r, pivot)
5. if (k == pivot_Index)
6. return A[k]
7. else if (k < pivot_Index)
8. quickselect (A, l, pivot_Index -1, k)
9. else
10. quickselect (A, pivot_Index+1, r, k)

B). In random quickselect algorithm, the pivot is randomly selected at each time which is uniformly distributed between 1 and N (where N is the length of input array). So, we can say there is at least ½ chance to select the random pivot between $\frac{N}{4}$and $\frac{3N}{4}$. This means at least half of the time, the problem can be reduced to no more than ¾ of the previous size. And let's say the pivot lies in $[\frac{N}{4}, \frac{3N}{4}]$ is a good pivot.

Based on such analysis, we could expect the problem is reduced to $\frac{3N}{4}$ after two rounds. Let T(n,s) be the runtime random variable. $T(n,s) \leq T(\frac{3n}{4}, s) + X(s) \cdot dn$ where $X(s)$ is times to find a good pivot and dn is runtime of partition.

$$\text{E}(T(n,s)) \leq E(T(\frac{3n}{4}, s)) + E(X(s) \cdot dn)$$

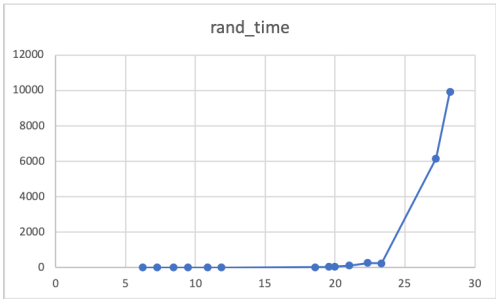
$$\text{E}(T(n,s)) \leq E(T(\frac{3n}{4}, s)) + E(X(s)) \cdot dn$$

$$\text{E}(T(n,s)) \leq E(T(\frac{3n}{4}, s)) + 2 \cdot dn$$

$$T_{exp}(n) \leq T_{exp}(\frac{3n}{4}) + 2 \cdot dn$$
$$T_{exp}(n) \leq n((\frac{3}{4}) + (\frac{3}{4})^2 + \ldots) + 2 \cdot dn \; s = O(n)$$

So the average time complexity of the random quickselect algorithm is O(n).

D). Seven more tests are generated for testing. The following graphs' x is logn where n is the size of array, y is the program running time.



Random_actual_graph

Since the x-axis is logn, the graph is compared with $y = e^x$. The actual graph is very similar to the expected graph which shows the code's time complexity is O(n).

4. Yes, the deterministic algorithm can solve above problem in $\theta(n)$ Algorithm:
   1. Divide n elements into $\lfloor \frac{n}{5} \rfloor$ and each group have 5 elements and at most one group has n mod 5 elements
   2. Find the median of groups by insert-sorting firstly then pick the median from the sorted list
   3. Recursively find all groups' median and find the median of all found median (if there are even number of medians, use the lower median) (SELECT)
   4. Partition the input array. k elements in the low side, x is the k-th smallest element and n-k elements in the high side.
   5. If i = k, return x. If i < k, call SELECT on the low side. If i > k, call SELECT on the high side.

   Time complexity:
   Step 1, 2, 4 all take O(n).
   Step 3 takes $T(\lceil n/5 \rceil)$ and step 5 at most takes $T(\lfloor 7n/10 \rfloor)$

$$T(n) \leq \begin{cases} O(n) & if\ n < N \\ T(\lceil n/5 \rceil) + T(\lfloor 7n/10 \rfloor) & otherwise \end{cases}$$

$$T(n) \leq T(\lceil n/5 \rceil) + T(\lfloor 7n/10 \rfloor) + O(n)$$
$$= T(\lceil n/5 \rceil) + T(\lfloor 7n/10 \rfloor) + cn \; (where\ c\ is\ a\ constant)$$
(1/5 + 7/10 = 9/10)
$$= cn(1 + (9/10) + (9/10)^2 + (9/10)^3 + \cdots)$$
$$= \theta(n)$$

Reference:

1. https://en.wikipedia.org/wiki/Binary_search_algorithm (PartA 3)

2. https://www.quora.com/How-is-time-taken-by-a-randomized-quick-select-algorithm-in-O-n (PartB 3)
3. Discussed with Chaahat Jain(u6398806) and Simian Wang(u6165791)