1. $T(n) = aT(\frac{n}{b}) + f(n)$

(a) $T(n) = 7T(\frac{n}{2}) + n^2$

$\because a=7, b=2, f(n) = n^2$

$\log_b a = \log_2 7 \approx 2.807$

$f(n) = O(n^{\log_2 7 - \varepsilon})$ $(\varepsilon \approx 0.807)$

$\therefore T(n) = O(n^{\log_2 7}) \approx O(n^{2.807})$

$\therefore$ It's case 1.

(b) $T(n) = T(\frac{n}{2}) + 1$

$\because a=1, b=2, f(n) = 1$

$\log_b a = \log_2 1 = 0$

$f(n) = O(n^0) = O(n^{\log_2 1})$

$\therefore T(n) = O(n^{\log_b a} \log^{k+1} n) =$

$O(n^0 \log^{0+1} n) = O(\log n)$ $(k=0)$

$\therefore$ It's case 2.

(c) $T(n) = 4T(\frac{n}{2}) + n^3$

$\because a=4, b=2, f(n) = n^3$

$\log_b a = \log_2 4 = 2$

$f(n) = n^3 = \Omega(n^{2+\varepsilon})$ $(\varepsilon = 1)$

$af(\frac{n}{b}) = 4 \cdot (\frac{n}{2})^3 = 4 \cdot \frac{n^3}{8} = \frac{n^3}{2} \le cn^3$

$(c = \frac{1}{2} < 1)$

$\therefore T(n) = O(n^3)$ $\therefore$ It's case 3.

---

2. Algorithm FindLightQuarter (quarters):

```
Input: quarters [1 to 81]
Output: the index of the light quarter
current_group = quarters
for i=1 to 4:
    n= length(current_group)
    group_size = n/3
    group1 = current_group[1 to groupsize]
    group2 = current_group[groupsize+1 to 2*groupsize]
    group3 = current_group[2*groupsize+1 to n]
    weigh group1 vs group2
    if group1 == group2:
        current_group = group3
    else if group1 < group2:
        current_group = group1
    else:
        current_group = group2
return current_group[only 1]
```

---

3. Algorithm TopTwoCandidates (votes):

```
Input: votes [1 to n]
Output: top_two, each_count_gt_half
if n==1:
    vote = votes[1]
    counts = empty_map
    counts[vote.candidate1] = 1
    counts[vote.candidate2] = 1
    return top_two = [vote.candidate1, vote.candidate2],
        each_count_gt_half = [False, False]
mid = n/2
left_top, left_flag = TopTwoCandidates(votes[1tomid])
right_top, right_flag = TopTwoCandidates(votes
                                        [mid+1 ton])

counts = empty_map
for candidate in left_top:
    counts[candidate] += count_votes(candidate,
                                     votes[1tomid])
for candidate in right_top:
    counts[candidate] += count_votes(candidate,
                                     votes[mid+1 to n])
sorted_candidates = sort_by_count_descending(counts)
top_two = sorted_candidates[1to2]
each_count_gt_half = [counts[top_two[0]] > n/2,
                      counts[top_two[1]] > n/2]
return top_two, each_count_gt_half
```

---

4. (a) $p = \frac{1}{n!}$

(b) $p = \frac{1}{n!}$

(c) 1 2 3. ⋯ n. $p = \frac{n-1}{n!} = \frac{1}{n(n-2)!}$

---

5. For each element i

$X_i = \begin{cases} 1 & \text{if it stays in previous position} \\ 0 & \text{otherwise} \end{cases}$

For each element. the probability that it staysin each previous position is $\frac{1}{n}$

$\therefore E(X) = E(X_1 + X_2 + \cdots + X_n) =$

$E(X_1) + E(X_2) + \cdots + E(X_n)$

$E(X_i) = 1 \cdot \frac{1}{n} + 0 \cdot \frac{n-1}{n} = \frac{1}{n}$

$\therefore E(X) = \sum_{i=1}^{n} E(X_i) = n \cdot \frac{1}{n} = 1$