American University of Beirut
Department of Computer Science
CMPS 211 - Spring 2015-16
Assignment 6
Maximum Grade is 120

The following exercises are to be completed by Monday 2nd of May. Late submissions will not be accepted.
You are expected to submit to moodle and another printed copy during class. Only typed submissions will be accepted from now on, and those who submit in LaTeX will get extra credits.

Exercise 1 (15 Points)

Prove that among any 7 points placed on the circumference of any circle, there is always a pair of points that form an arc of angle ≤ 60 degrees?

Solution

Split the circle into 6 arcs each of an angle of 60 degrees. Now, we have transformed the problem into a pigeonhole problem having k=6 pigeon holes (6 arcs), and N=7 pigeons (the 7 points). By the pigeon hole principle, at least two points fall into the same arc. Since all arcs are of size 60 degrees, then angle between the two points which fall in the same arc is at most 60 degrees.

Therefore, among any 7 points placed on the circumference of any circle, there is always a pair of points that form an arc of angle ≤ 60 degrees.

Exercise 2 (15 Points)

A coin is flipped 10 times, where each flip comes up either heads or tails. How many possible outcomes do we have in these cases:

1. In total?
   *Solution:*We have 10 flips, each flip has two possible outcomes, therefore the product rule applies: 2^{10}.

2. Contain exactly two heads?
   *Solution:*We have 10 flips, 2 of them must be head, the rest is automatically chosen to be tails. Since the faces of the coins are identical then ordering the heads doesn't matter (i.e. $H_1H_2T_1...$ is equivalent to $H_2H_1T_1...$). Therefore the total number of outcomes is equal to $10C2$. We can see it as follows: We have 2 Heads which we want to pick 2 position out of 2. The position order doesn't matter; therefore pick 2 out of 10 order doesn't matter & without replacement, then $10C2$.

3. Contain at most three tails?
   *Solution:*This is equal to the sum of outcomes containing exactly no Tails, exactly one Tail, exactly two Tails, and exactly three Tails (same logic as in part b): $10C0 + 10C1 + 10C2 + 10C3$.

4. Contain the same number of heads and tails?
   *Solution:*This is equivalent to saying exactly 5 heads (or exactly 5 tails). Again, we can apply the same logic as in part b, so $10C5$.

Exercise 3 (15 Points)

How many bit strings of length 20 contain:

1. Exactly 4 ones?
   *Solution:*This is similar to problem 2-2, in fact if we consider 1 to be tail and 0 to be heads it will be exactly mappable to it. In this case we want 4 ones so it is $20C4$

2. At most 4 ones?
   *Solution:*Also similar to problem 2-3 Sum of exactly 0 ones, exactly 1 one, exactly 2 ones, exactly 3 ones, and exactly 4 ones... Therefore $20C0 + 20C1 + 20C2 + 20C3 + 20C4$

3. At least 4 ones?
   *Solution:*This can be computed in many ways, we can apply the sum rule directly and get $\sum_{n=4}^{20} 20Cn$, or we can think about it as if it is saying at most 16 ones and apply the sum rule then $\sum_{n=0}^{16} 20Cn$. Or we can think about it as the complement of having at most 3 ones and thus we get $2^{20} - (20C0 + 20C1 + 20C2 + 20C3)$.

4. An equal number of zeros and ones?
   *Solution:*In other word exactly 10 zeros (or exactly 10 ones): $20C10$.

5. Either 10 consecutive zeros or 10 consecutive ones?
   Solution: First we count the number of bit strings of length 20 that contain 10 consecutive 0s. We will base the count on where the string of 10 or more consecutive 0s starts. If it starts in the first bit, then the first 10 bits are all 0s, but there is free choice for the last 10 bits; therefore there are 2^10 such strings. If it starts in the second bit, then the first bit must be a 1, the next 10 bits are all 0s, but there is free choice for the last 9 bits; therefore there are 2^9 such strings. If it starts in the third bit, then the second bit must be a 1 but the first bit and the last 8 bits are arbitrary; therefore there are 2^9 such strings. Similarly, there are 2^9 such strings that have the consecutive 0s starting in each of positions four, five, ... and eleven. This gives us a total of $2^10 + 10 * 2^9$ strings that contain 10 consecutive 0s. Symmetrically, there are $2^10 + 10 * 2^9$ strings that contain 10 consecutive 1s. Clearly the two strings 00000000001111111111 and 11111111110000000000 are counted twice, and therefore should be deducted. Therefore the answer is $2 * (2^10 + 10 * 2^9) - 2$

Exercise 4 (15 Points)

How many string of six lowercase letter from the English alphabet contain:

1. The letter a?
   *Solution:*The easiest way to do this is counting the strings not containing a, and take that out all possible strings: $26^6 - 25^6$. Choosing a position for a then counting can be tricky and cause overcounting for the cases where we have two consecutive letters a.

2. The letters a and b?
   *Solution:*In a similar way we count the number of string not containing b (25^6), and the strings not containing a (25^6), and the strings not containing neither a nor b (24^4). We notice that the strings not containing a include the strings not containing neither a nor b, and those not containing b also contain the ones without a and b; and therefore the strings not containing a or b are overcounted, and should be subtracted : $25^6 + 25^6 - 24^6$

   The total number of strings with both a and b is the total number of strings that can formed from the alphabets minus those without a and b, and therefore: $26^6 - (25^6 + 25^6 - 24^6)$.

3. The letters a and b in consecutive positions, with a preceding b, with all the letters distinct?
   *Solution:*In other words we want the string to contain "ab", and the string has distinct letters (no need to worry about double counting).

   This leave 4 free letters to pick out of 24, since we can't pick a nor b, and their order matters), so we get $4P24 = 24 * 23 * 22 * 21$ different such strings. We then multiply this number by 5 which is the number of positions we can insert "ab" into, overall we get $5 * 4P24$.

4. The letters a and b, where a is somewhere to the left of b in the string, with all the letters distinct?
   *Solution:*We have 4 free letters similar to the previous part ($24P4$ possible permutation of those letters), we can place a anywhere (5 possibilities), however we can place b only to the right of a (number of possibilities depends on position of a).

   When a is placed in the beginning, b can be placed any where after it (5 positions), When a is placed at the end, b can only be placed at the end of the string after a (1 position). So we get $5 + 4 + 3 + 2 + 1 = 15$ different ways of placing a and b inside the string.

   This gives us overall $15 * 24P4$.

Exercise 5 (15 Points)

What is the minimum number of students required in a discrete mathematics class to be sure that at least six will receive the same grade, if there are five possible grades, A, B, C, D, and F?

Solution

We have 5 possible grades, and we need to guarantee at least 6 students take same grade, therefore we want to find N such that $\lceil \frac{N}{5} \rceil = 6$ by pigeon hole principle; therefore we know that $5 * (6 - 1) + 1 \leq N \leq 5 * 6$, and hence the minimum N to satisfy the equation is 5*(6-1) + 1 = 26 students.

Exercise 6 (20 Points)

You are given an array of n elements. Devise a divide and conquer algorithm to remove all duplicates in array. Prove the correctness of your algorithm. Then write and solve a recurrence for the algorithm's run-time.

Algorithm 1 $merge - sort - no - dups(a_0, ..., a_n, lower, upper)$

```
1: if lower == upper then
2:    return a_lower
3: else if lower < upper then
4:    left = merge-sort-no-dups(a_0, ..., a_n, lower, (upper+lower)/2)
5:    right = merge-sort-no-dups(a_0, ..., a_n, (upper+lower)/2, upper)
6:    return merge-no-dups(left, right)
7: end if
```

Algorithm 2 $merge - no - dups(l_0, ..., l_m, r_0, ..., r_k)$

```
1: result = empty list
2: i = 0 j = 0
3: while i ≤ m ∧ j ≤ k do
4:    if l_i = r_j then
5:       add l_i to end of result
6:       i = i + 1
7:       j = j + 1.
8:    else if l_i < r_j then
9:       add l_i to end of result
10:      i = i + 1
11:   else if l_i > r_j then
12:      add r_j to end of result
13:      j = j + 1
14:   end if
15: end while
16: while i ≤ m do
17:   add l_i to end of result
18:   i = i + 1
19: end while
20: while j ≤ k do
21:   add r_j to end of result
22:   j = j + 1
23: end while
24: return result
```

Correctness

We want to show the following: *merge-sort-no-dups($a_0, ..., a_n$, lower, upper) return a sorted duplicate-free copy of the segment $a_{lower}, ..., a_{upper}$.* call this statement $P(lower, upper)$.

We will proceed by induction, first we show that P is true in the base case, i.e. when $lower = upper$. This is clearly the case since the algorithm will return a single element.

For the inductive step we will look at $P(lower, upper)$ for some $upper > lower$, here we notice that the algorithm makes two recursive calls, on the first and last half of $a_{\text{lower}}, ..., a_{\text{upper}}$, these calls are assumed to be correct by induction hypothesis, therefore we know that both left, and right are sorted duplicate free segments of the first and last half of $a_{\text{lower}}, ..., a_{\text{upper}}$.

Therefore the correctness depends on the correctness of merge-no-dups. We need to show that merge-no-dups merges left and right into a sorted duplicates free array. For this purpose we suggest the following loop invariant: *inv = "At iteraton (i, j) result is sorted and duplicate free of the segments $l_0, ..., l_{i-1}$ and $r_0, ..., r_{j-1}$"*. Where l and r are the same as left and right previously discussed.

Initialization
At the start of the loop, $i = j = 0$ and result is empty, so the invariant holds trivially.

Maintenance
At the start of the (i, j) iteration, we assume that the invariant holds, we want to show that it holds after the end of the iteration. We have three cases:

- $l_i = r_j$: in this case we add only one of the two numbers, leaving the other out, then we proceed by increasing i and j. since l, r are sorted and duplicate free, as well as result (by hypothesis), then $l_i = r_i > t \ \forall t \in result$. Therefore result remains sorted and duplicate free and now includes l_i, r_j.

- $l_i < r_j$: in this case we add l_i and move i forward, since l is sorted, therefore $l_i > l_k \ \forall 0 \leq k < i$, also r and result are sorted and duplicate free, therefore $l_i > t \ \forall t \in result$. Therefore result remains sorted and duplicate free and now includes l_i.

- $l_i > r_j$: in this case we add r_j and move j forward, since r is sorted, therefore $r_j > r_k \ \forall 0 \leq k < j$, also l and result are sorted and duplicate free, therefore $r_j > t \ \forall t \in result$. Therefore result remains sorted and duplicate free and now includes r_j.

Termination
The loop clearly terminates when either i or j exceeds the size of l or r. The two consecutive loops clearly wont change the invariant, since both list are sorted and duplicate, the remaining segment in either of them, is guaranteed to be strictly greater than the previous segment.

Therefore, merge-no-dups returns a sorted duplicate free copy of l merged with r, and thus our algorithm is correct.

Run-time

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

By Master theorem we can find that run-time is $O(nlgn)$.

Exercise 7 (15 Points)

Write the recurrence relation describing the run-time of each of the following recursive algorithms

7.1 Ternary Search (5 Points)

int ternary_search(**int list**: $a_1, a_2, ..., a_n$, **int** key, **int** left, **int** right):
    If left > right
        return 0

    $third_1 := \lfloor \frac{left+right}{3} \rfloor$
    $third_2 := \lfloor 2 \times \frac{left+right}{3} \rfloor$

    if key = a[$third_1$]
        return $third_1$

    else if key = a[$third_2$]
        return $third_2$

    else if key < a[$third_1$]
        right := $third_1 - 1$

    else if key > a[$third_2$]
        left := $third_2 + 1$

    else
        left := $third_1 + 1$
        right := $third_2 - 1$

    return ternary_search($a_1, a_2, ..., a_n$, key, left, right);

This is usually initially called as on left = 1, right = n.

7.1 Run-time

$$T(n) = T(\frac{n}{3}) + O(1)$$
$$T(1) = C$$
$$\in \Theta(log_3(n))$$

7.2 Linear Search (5 Points)

int linear_search(**int list**: $a_1, a_2, ..., a_n$, **int** key, **int** index):
    if index < 0
        return -1;
    else if key = a[index]
        return index
    else
        return linear_search($a_1, a_2, ..., a_n$, key, index -1)

This is initially called on index = n.

7.2 Run-time

$$T(n) = T(n-1) + O(1)$$
$$T(1) = c$$

We can use the iteration method:

$$T(n) = T(n-1) + c'$$
$$= T(n-2) + c' + c'$$
$$= T(n-3) + c' + c' + c'$$
$$...$$
$$= T(n-i) + i \times c'$$

We stop iterating/recursing when we hit the base case (at $n = 1, i = n - 1$). at which case we get $T(n - n + 1) + (n - 1)c' = c + (n - 1)c' \in \Theta(n)$

7.3 Max/Min (5 Points)

int max_min(**int list**: $a_1, a_2, ..., a_n$, **int** key, **int** index):
    if n = 2
        if $a_1 \geq a_2$
            return (a_1, a_2)
        else
            return (a_2, a_1)
    else if n = 1
        return (a_1, a_1)

S1 := $\{a_1, a_2, ..., a_{\frac{n}{4}}\}$
$S_2 := \{a_{\frac{n}{4}+1}, a_{\frac{n}{4}+2}, ..., a_{\frac{n}{2}}\}$
$S_3 := \{a_{\frac{n}{2}+1}, a_{\frac{n}{2}+2}, ..., a_{3\times\frac{n}{4}}\}$
$S_4 := \{a_{3\times\frac{n}{4}+1}, a_{3\times\frac{n}{4}+2}, ..., a_n\}$

(max_1, min_1) := max_min(S_1)
(max_2, min_2) := max_min(S_2)
(max_3, min_3) := max_min(S_3)
(max_4, min_4) := max_min(S_4)

maxs = [max_1, max_2, max_3, max_4]
mins = [min_1, min_2, min_3, min_4]

(max, __) := max_min(maxs) # __ **means value is ignored**
(__, min) := max_min(mins)

return (max, min)

7.3 Run-time

Notice that max_min(maxs) and max_min(mins) run on arrays of fixed constant size (4), therefore they are considered to run in constant time $O(1)$.

$$T(n) = 4T(\frac{n}{4}) + O(1)$$
$$T(1) = c$$
$$\in \Theta(n)$$

Exercise 8 (10 Points)

Use Masters theorem to run-time of each of the following recurrences:

1. $T(n) = 2T(\frac{n}{2}) + O(n^2)$.
   $log_b(a) = log_2(2) = 1$. clearly $n^2 \in \Omega(n^{1+\epsilon})$ ($\epsilon < 1$).
   Also $2f(\frac{n^2}{2}) \leq 2n^2$ for all n (Regularity condition holds).
   Therefore $T(n) \in \Theta(n^2)$

2. $T(n) = T(\frac{n}{2}) + O(1)$.
   $log_b(a) = log_2(1) = 0$. Notice that $n^{log_b(a)} = n^0 = 1$ Constant.
   Therefore $f(n) \in \Theta(n^{log_b(a)})$.
   Therefore $T(n) \in \Theta(n^{log_b(a)} log(n)) \in \Theta(log(n))$

3. $T(n) = 2T(\frac{n}{2}) + O(1)$.
   $log_b(a) = log_2(2) = 1$. Notice that $f(n)$ is constant,

Therefore it is in $O(n^{log_b(a)-\epsilon})$ $(\epsilon < 1)$.
Therefore $T(n) \in \Theta(n^{log_b(a)}) \in \Theta(n)$

4. $T(n) = 4T(\frac{n}{2}) + O(n^3)$.
   $log_b(a) = log_2(4) = 2$. Notice that $n^3 \in \Omega(n^{2+\epsilon})$ $(\epsilon < 1)$.
   Also $4\frac{n^3}{2} \leq 4n^3$ for all n (Regularity condition holds).
   Therefore $T(n) \in \Theta(n^3) \in \Theta(n^2)$

5. $T(n) = 2T(\frac{n}{2}) + 7n$.
   $log_b(a) = log_2(2) = 1$. Notice that $f(n) = n \in \Theta(n^{log_b(a)})$.
   Therefore $T(n) \in \Theta(n^{log_b(a)} log(n)) \in \Theta(nlog(n))$