

Question 1. With sequencing by synthesis, are errors more likely at the beginning of the read, in the middle, or at the end, or equally likely throughout? Why?

When we are sequencing by synthesis, the sequencing error are more likely occur at the **end** of the throughout.

It is because that we determine the sequencing reads by sequencing cycles and a cluster of stands for each read. If a strand cannot be terminated by its terminator for one sequencing cycle, then it will fall out of sync. Therefore, the number of strands that fall out of sync is proportional to the number of sequencing cycles. As the number of cycles increases, more strands could fall out of sync, which means they are glowing wrong colors. At the end of the throughout, the number of strands that fall out of sync would be the biggest, so the base color of a cluster will more likely be ambiguous compare to the beginning and the middle. Therefore, the probability of occurring a sequencing error will be bigger.

Question 2. What are the pros and cons of short and long reads?

Advantages of short reads:

1. Easier to implement
2. Cheaper to produce
3. Completely adequate when we have a reference genome

Disadvantages of short reads:

1. Short reads are suffered from repetitive sequences, which can cause problems such as over collapsed
2. Isoforms and gene fusions are hard to be detected
3. Trouble discriminating paralogous sequences
4. Difficulties in phasing alleles

Advantages of long reads:

1. Able to span repetitive regions
2. More powerful for sequence assembly (no reference genome)
3. Ability of identify large structural variations
4. Produce better genome assembly quality

Disadvantages of long reads:

1. Making the Long reads is a hard-technical problem
2. Much more expensive to generate
3. Higher error rates
4. Reduced quantification abilities

Question 3. Write pseudo-code for Boyer-Moore and give an example (not the same as Ben's) of how it works.

Pseudo-code for Boyer-Moore:

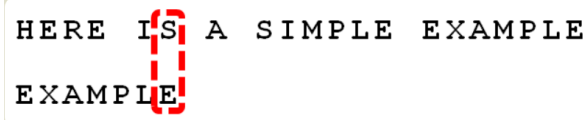
Input: Pattern string P, target string T.

Output: return the starting index of the matched substring in text T.

```
i = 0;
while(i <= T.length()-P.length()) {
    Skips_num = 0;
    for(j = P.length()-1; j >= 0; j--) {
        if(P.charAt(j) != T.charAt(i + j)) {
            if(mismatchShiftsTable.get(T.charAt(i + j)) != null) {
                Skips_num = mismatchShiftsTable.get(T.charAt(i+j));
                break;
            }else {
                Skips_num = P.length();
                break;
            }
        }
    }
    i += Skips_num;
    if(Skips_num == 0) {
        return i;
    }
}
```

Example of how it works:

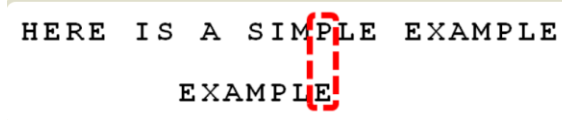
1. First, string P and tring T are aligned at the head, and the comparison starts from the tail of string P.



```
HERE IS A SIMPLE EXAMPLE
EXAMPLE
```

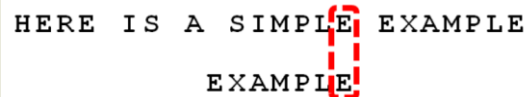
Since 'S' is not matching with 'E', 'S' is the bad character here. Also, since 'S' has never occurred in string P, Shift Index = $6 - (-1) = 7$

2. Start the matching from tail again, since 'P' is not matching with 'E', 'P' is the bad character here. However, 'P' occurs at index 4 of string P, Shift Index = $6 - 4 = 2$



```
HERE IS A SIMPLE EXAMPLE
      EXAMPLE
```

3. Again, we match from the tail, and we have a match of 'E' and 'E' now.



HERE IS A SIMPLE EXAMPLE
EXAMPLE

Characters in string P and string T keep matching until 'I' in T and 'A' in P.



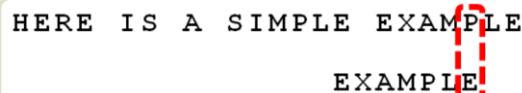
HERE IS A SIMPLE EXAMPLE
EXAMPLE

Hence, 'I' is the bad character now, $\text{Shift Index} = 2 - (-1) = 3$ under Bad Character rule.

However, since we have a match of "MPLE", which is a good suffix here, we can also calculate the shift index under Good Suffix rule. $\text{Shift Index} = 6 - 0 = 6$.

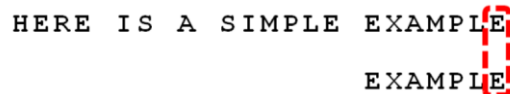
And $6 > 3$, so we shift 6 index using good suffix rule.

4. Next, we find a bad character 'P' again, and the $\text{Shift Index} = 6 - 2 = 4$.



HERE IS A SIMPLE EXAMPLE
EXAMPLE

5. Finally, starting from the tail, it is compared bit by bit, and all matches are found, so the search ends.



HERE IS A SIMPLE EXAMPLE
EXAMPLE

If we need to continue the search (that is, find all matches), according to the good suffix rule, $\text{Shift Index} = 6 - 0 = 6$, which means move the 'E' at the head to the position of the 'E' at the tail.

Question 4. How is the Pigeonhole Principle used in approximate matching with an index for exact matching?

Pigeonhole Principle states that if there are n pigeons and $n + 1$ pigeonholes, then there must appear an empty pigeonhole. In general, pigeonhole principle converts approximate matching algorithm to exact matching algorithm. More specifically, If the pattern P occurs in text T with up to k edits (different base), then we can chop P into $k + 1$ partitions. At least one of these partitions must appear with 0 edit. We perform any exact matching algorithm with partitions and text T . At least one of the partitions will have a match with text T . Then we verify the neighborhoods of the matched partition to determine whether the entire string P occurs in these neighborhoods. If the neighborhoods and the matched partition together match the pattern P , then there is a match of text T and pattern P . Otherwise, we look at the next matched partition with the same process until there is no matched partition, which means text T does not have a match with pattern P .

Question 5. For the final cost matrix Ben presented for approximate matching, write a sentence explaining each distinct cost.

Final cost matrix:

	A	C	G	T	-
A	0	4	2	4	8
C	4	0	4	2	8
G	2	4	0	4	8
T	4	2	4	0	8
-	8	8	8	8	

First , we need to illustrate some details of four bases.

1. A and G both belong to purines
2. C and T both belong to pyrimidines
3. Transitions: from same category to same category (A to G; C to T; G to A; T to C)
4. Transversions: all other substitutions

Cost of 0: It occurs when the conversion is between two same bases such as A to A, C to C. Since we do not need to change anything, the cost would be 0.

Cost of 2: It occurs when the conversion is a transition. Since the transitions is between same category; also, it occurs more often than transversions and deletion or insertion. Hence, it has a lower cost, which is 2.

Cost of 4. It occurs when the conversion is a transversion. Since the transversion is between different categories, also, it occurs half frequent as transition. Hence, it has a double cost of transversion, which is 4.

Cost of 8: It occurs when the conversion is an insertion or deletion. Since the process is converting a base to nothing or converting nothing to a base, it would cost the most. Also, the occurrence of substitution is about 1/1000, and the occurrence of gap about is 1/3000. Hence, it has a highest cost, which is 8.