Question 1

1. Based on this condition, we have:

$$f_1(j) = \begin{cases} a_{1,1} & j = 1 \\ min(f_1[j-1] + a_{1,j}, f_2[j-1] + t_{2,j-1} + a_{1,j}) & j > 1 \end{cases}$$

$$f_2(j) = \begin{cases} a_{2,1} & j = 1 \\ min(f_2[j-1] + a_{2,j}, f_1[j-1] + t_{1,j-1} + a_{2,j}) & j > 1 \end{cases}$$

$$f^* = min(f_1[j], f_2[j])$$

So:

$$f_{1,1} = 2$$
$$f_{2,1} = 4$$
$$f_1 = 2$$
$$S \to S_{1,1}$$

$$f_{1,2} = min(2+6, 4+1+6) = 8$$
$$f_{2,2} = min(2+2+5, 4+5) = 9$$
$$f_2 = 8$$
$$S \to S_{1,1} \to S_{1,2}$$

$$f_{1,3} = min(8+4, 9+2+6) = 12$$
$$f_{2,3} = min(8+4+8, 9+8) = 17$$
$$f_3 = 12$$
$$S \to S_{1,1} \to S_{1,2} \to S_{1,3}$$

$$f_{1,4} = min(12+8, 17+1+8) = 20$$
$$f_{2,4} = min(12+1+4, 17+4) = 17$$
$$f_4 = 17$$
$$S \to S_{1,1} \to S_{1,2} \to S_{1,3} \to S_{2,4}$$

	$f_1[j]$ $(line)$	$f_2[j]$ $(line)$
1	2 (S)	4 (S)
2	8 (1)	9 $(1 \to 2)$
3	12 (1)	17 (2)
4	20 (1)	17 $(1 \to 2)$

2. The minimum time is 17.

3. Starting from the end, we always choose the minimum value and track back which line from. For example, from $f_{2,4}$ backward is line 1, and from $f_{1,3}$ backward is also line 1.

   So the fastest way is $S_{1,1} \to S_{1,2} \to S_{1,3} \to S_{2,4}$.

Question 2

1. Shift table:

Letter	A	G	C	T
Value	5	1	4	2

2.

A	G	C	C	G	T	G	C
C	G	T	G	C			

A	G	C	C	G	T	G	C
	C	G	T	G	C		

A	G	C	C	G	T	G	C
			C	G	T	G	C

   The comparison count is $1 + 1 + 5 = 7$.

Question 3

1. Loop 1

	a	b	c	d	e
	0	∞	∞	∞	∞

Loop 2

	a	b	c	d	e
	0	−1	5	4	−3

ac(5)	0	∞	5	∞	∞
ae(4)	0	∞	5	∞	4
ce(-8)	0	∞	5	∞	−3
eb(2)	0	−1	5	∞	−3
ed(7)	0	−1	5	4	−3

ac(5)	0	∞	5	∞	∞
ae(4)	0	∞	5	∞	4
ce(-8)	0	∞	5	∞	−3
eb(2)	0	−1	5	∞	−3
ed(7)	0	−1	5	4	−3

The loop 1 is equal to the loop 2, which means all vertices attain the minimum value. So the result is shown as the loop 2.

Question 4

1. Global alignment:

		A	A	T	G
	0	−5 ←	−10 ←	−15 ←	−20 ←
A	−5 ↑	2 ↖	−3 ↖←	−8 ←	−13 ←
G	−10 ↑	−3 ↑	−3 ↖	−8 ←	−6 ↖
C	−15 ↑	−8 ↑	−8 ↑	−8 ↖	−11 ↑

A A T G _

_ A _ G C

2. Local alignment:

		A	A	T	G
	0	0	0	0	0
A	0	2 ↖	2 ↖	0	0
G	0	0	0	0	2 ↖
C	0	0	0	0	0

A A G

A A G

Question 5

1. Class P includes all decision problems that can be solved within polynomial time in the worst-case scenario. Problems within P are deemed efficient because they can be resolved within polynomial time, which is considered an acceptable timeframe.
2. NP is the collection of all problems that can be confirmed or verified in polynomial time. Problems falling into “NP but not necessarily P” are those where solutions can be checked in polynomial time, yet no polynomial-time algorithms are currently known for solving them. A classic example is Sudoku; a 9x9 Sudoku is a P problem, but scaling it up to 99x99 Sudoku makes it no longer a P problem. NP problems often deal with extensive datasets and demand a significant amount of time to tackle, making solving them a more formidable task.
3. If a problem M is NP-hard, it means that every other problem in NP can be reduced to M in polynomial time. Additionally, if M is both in NP and NP-hard, it is classified as NP-complete (NPC). The significance of NP-completeness lies in the fact that if you can solve a problem in NPC, you can effectively solve all problems in NP since they can be reduced to M in polynomial time. This relationship highlights the critical role of NP-complete problems as benchmarks for the complexity of problems within the NP class.
4. When we say that decision problem A is polynomial time reducible to decision problem B, or that there exists a polynomial time reduction from A to B, it means that for any input a belonging to A, we can create an input b for B in polynomial time such that a is a “yes” instance if and only if b is a “yes” instance.

   The NP-completeness of a problem is proven by showing that it can be reduced to a known NP-complete problem using polynomial-time transformations. This establishes a link between the complexity of the new problem and the complexity of NP-complete problems, providing insights into its computational difficulty within the NP class.