Simon Fraser University

Department of Computer Science

Bioinformatics Algorithms - Assignment 3

Name: Niloufar Saeidi

ID: 301590708

5.1

In a maximization problem, approximation ratio = $min_{|\pi|=n}\frac{A(\pi)}{OPT(\pi)} = 4$. It means that the values of $\frac{A(\pi)}{OPT(\pi)}$ for different values of π are either greater than or equal to 4. Given that $A(\pi) = 12$, the values of $OPT(\pi)$ x are smaller than or equal to 3: $OPT \leq 3$.

5.4

- $\pi = 0|34|65|8|1|7|2|9$ the strip 6 5 is decreasing , $b(\pi) = 7$
- $\pi.\rho(1,4) = 0|3456|8|1|7|2|9$, $b(\pi) = 6$
- $\pi.\rho(1,4).\rho(7,8) = 0|3456|87|12|9, b(\pi) = 4$
- $\pi.\rho(1,4).\rho(7,8).\rho(6,7) = 0|345678|12|9, b(\pi) = 3$
- $\pi.\rho(1,4).\rho(7,8).\rho(6,7) = 0|345678|12|9, b(\pi) = 3$
- $\pi.\rho(1,4).\rho(7,8).\rho(6,7).\rho(1,6) = 0|876543|12|9, b(\pi) = 3$
- $\pi.\rho(1,4).\rho(7,8).\rho(6,7).\rho(1,6).\rho(8,9) = 0|87654321|9, b(\pi) = 2$
- $\pi.\rho(1,4).\rho(7,8).\rho(6,7).\rho(1,6).\rho(8,9).\rho(1,8) = 0123456789, b(\pi) = 0$

5.7

In a circular genome, we could say that using decreasing or increasing strips to identify which part to reverse is useless. Instead, we can develop a new reversal finding strategy:

Algorithm 1 Sorting by Reversals for circular genomes Algorithm

1: **procedure** ALG($a[]$)
2: **Note:** we take care of the fact that a[n-1] and a[0] are adjacent.
3: **while** $b(\pi) > 0$ **do**
4: find two values $m = a[i]$ and $n = a[j] = a[i] + 1$, $a[i] \in strip1$ and $a[j] \in strip2$, such that $strip1 \neq strip2$
5: find a substring s, such that if we reverse s, $strip1$ and $strip2$ become either one strip or neighboring strips.
6: reverse s
7: **if** m and n are not adjacent **then**
8: do a reversal such that they become adjacent
9: **end if**
10: **end while**
11: **end procedure**

In each iteration, we perform 1 or at most two reversals and reduce $b(\pi)$ by one. This means that $the\ number\ of\ reversals \leq 2b(\pi)$. Each reversal can at most reduce $b(\pi)$ by 2, therefore $d(\pi) = \frac{b(\pi)}{2}$. Therefore, $performance\ guarantee = \frac{2b(\pi)}{d(\pi)} = \frac{2b(\pi)}{\frac{b(\pi)}{2}} = 4$

5.9

According to the footer in page 127 of the textbook, by permutation we mean a permutation of consecutive integers. In this case, we can simply guess the position of each element of a sorted array is equal to the value of that element(we subtract it from one if we start the indices from zero). Therefore, a straightforward solution will be to swap each element arr[i] with the element sitting in its correct position arr[arr[i]-1]. Here's a pseudocode:

Algorithm 2 Optimal Swap Sorting Algorithm

```
1: procedure ALG(arr[])
2:     i ← 0
3:     while i ≠ len(arr) do
4:         correct_position ← arr[i] − 1
5:         if arr[i] ≠ arr[correct_position] then
6:             SWAP(arr[i], arr[correct_position])
7:         else
8:             i = i + 1
9:         end if
10:    end while
11:    return arr
12: end procedure
```

The number of swaps required is equal to the number of breakpoints plus one, which is the same as the number of cycles minus one. Each swap in this algorithm positions at least one element in its correct place. So we could claim that this is an optimal algorithm which minimizes the number of swaps.

5.13

breakpoints between $\pi^1 = 124356$ and $\pi^2 = 143256$: lets first extend them:
$\pi^1 = 01243567$ and $\pi^2 = 01432567$. 0 and 1 are adjacent to each other in π^2, so no breakpoints. 1 and 2 are not adjacent to each other in π^2, so (1,2) is a breakpoint. In the same way, (2,4), (3,5) are also breakpoints.

breakpoints between $\pi^1 = 124356$ and $\pi^3 = 0123465$:
lets first extend them:
$\pi^1 = 01243567$ and $\pi^3 = 01234657$. breakpoints are (2,4), (3,5), (6,7)

breakpoints between $\pi^2 = 143256$ and $\pi^3 = 0123465$:
lets first extend them:
$\pi^2 = 01432567$ and $\pi^3 = 01234657$. breakpoints are (1,4), (2,5), (6,7)