**Name:** Vishal Shashikant Salvi.

**UID:** 2019230069

**Class:** SE Comps

**Batch:** C

**Experiment No 8**

**Aim**: To implement the KMP algorithm for string Matching

**Theory**:

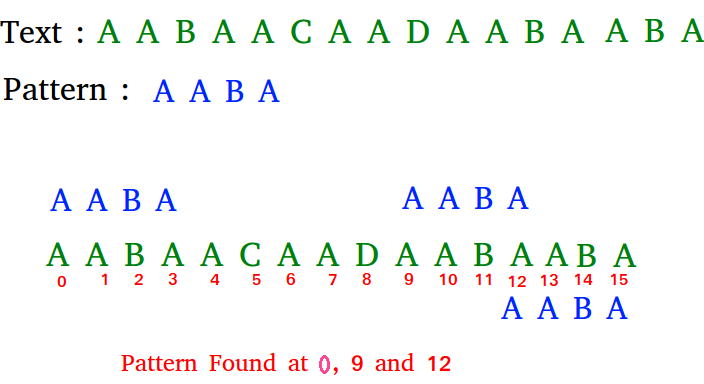
**The Knuth-Morris-Pratt (KMP)Algorithm**

Knuth-Morris and Pratt introduce a linear time algorithm for the string-matching problem. A matching time of O (n) is achieved by avoiding comparison with an element of 'S' that have previously been involved in comparison with some element of the pattern 'p' to be matched. i.e., backtracking on the string 'S' never occurs

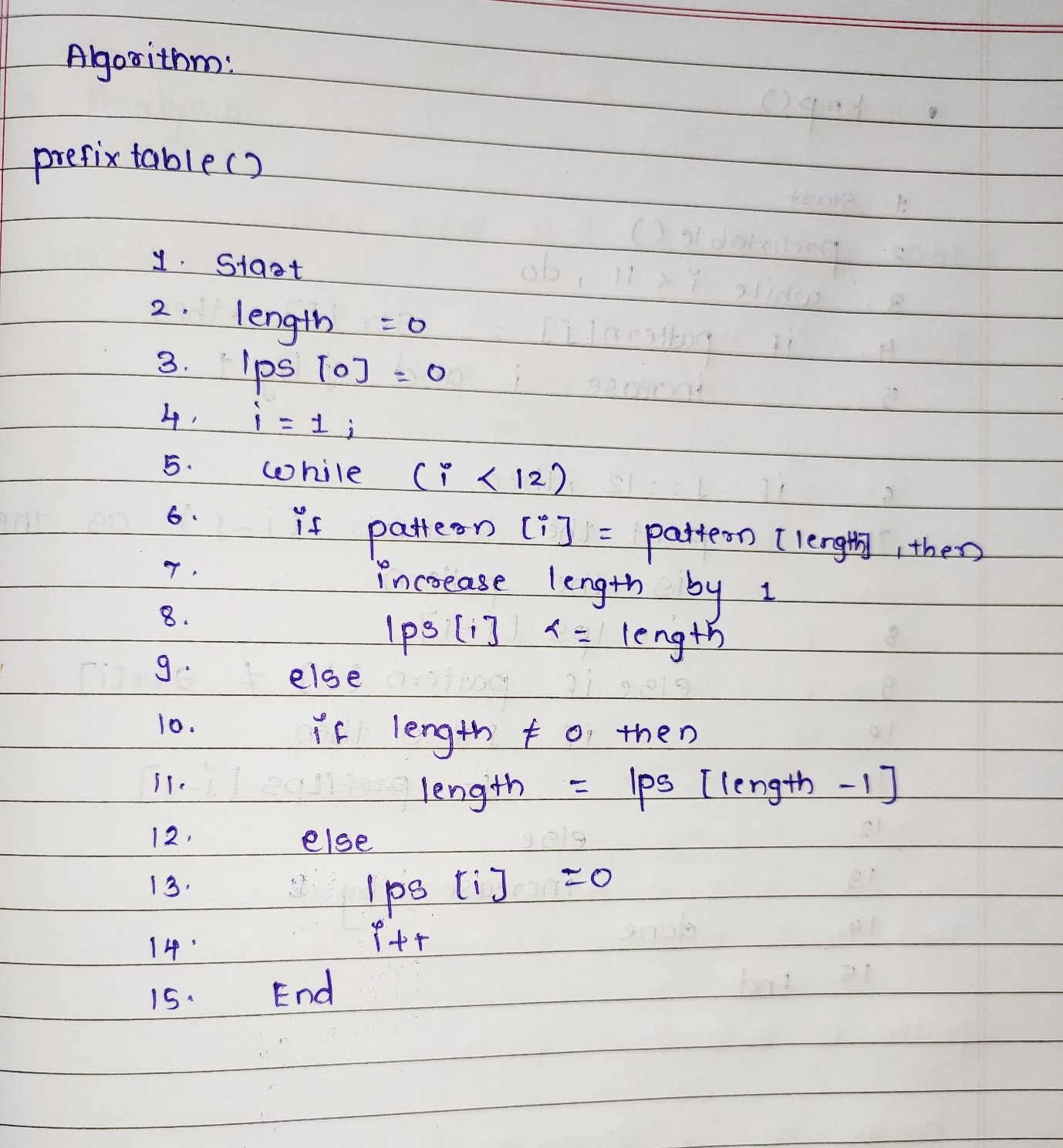
**Components of KMP Algorithm:**

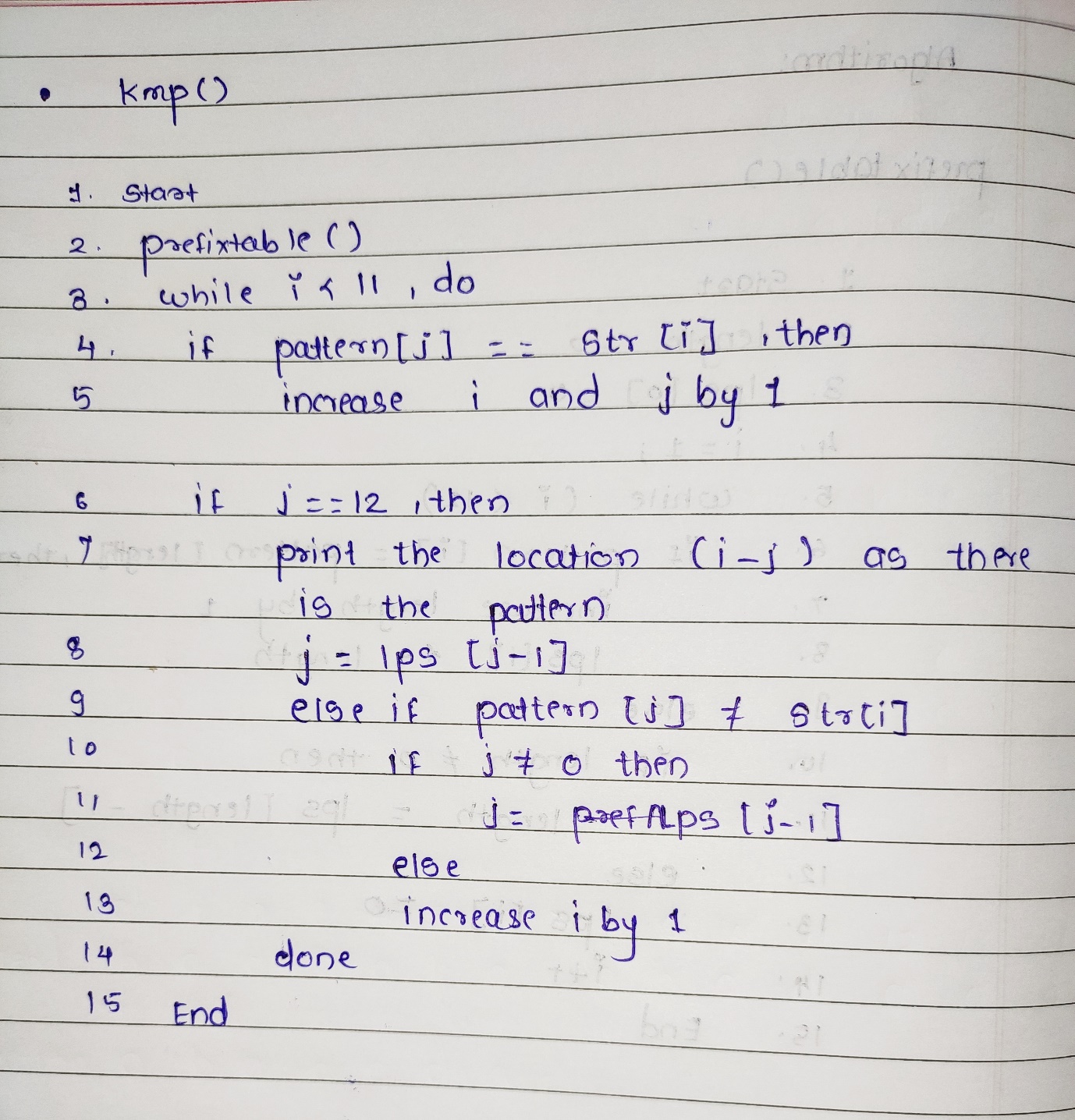
**1. The Prefix Function (Π):** The Prefix Function, Π for a pattern encapsulates knowledge about how the pattern matches against the shift of itself. This information can be used to avoid a useless shift of the pattern 'p.' In other words, this enables avoiding backtracking of the string 'S.'

**2. The KMP Matcher:** With string 'S,' pattern 'p' and prefix function 'Π' as inputs, find the occurrence of 'p' in 'S' and returns the number of shifts of 'p' after which occurrences are found.

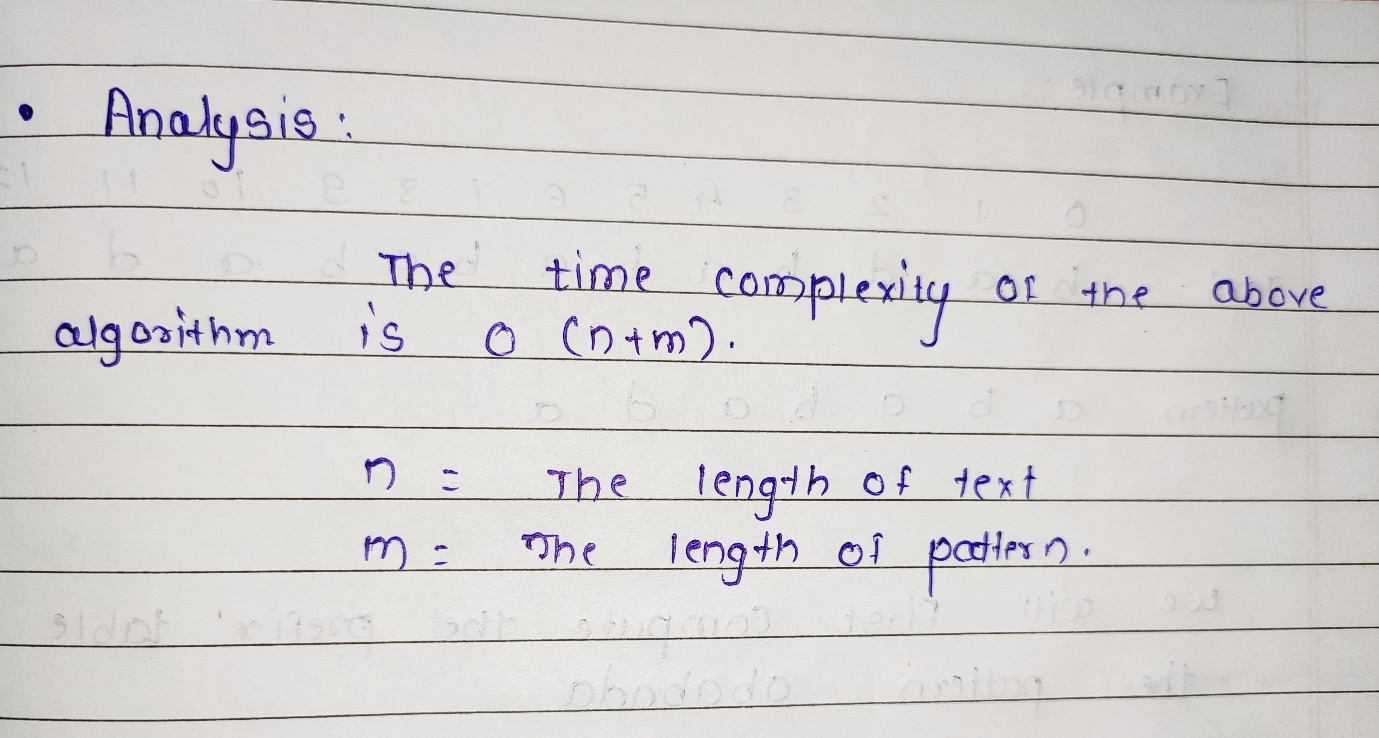


**Algorithm**:

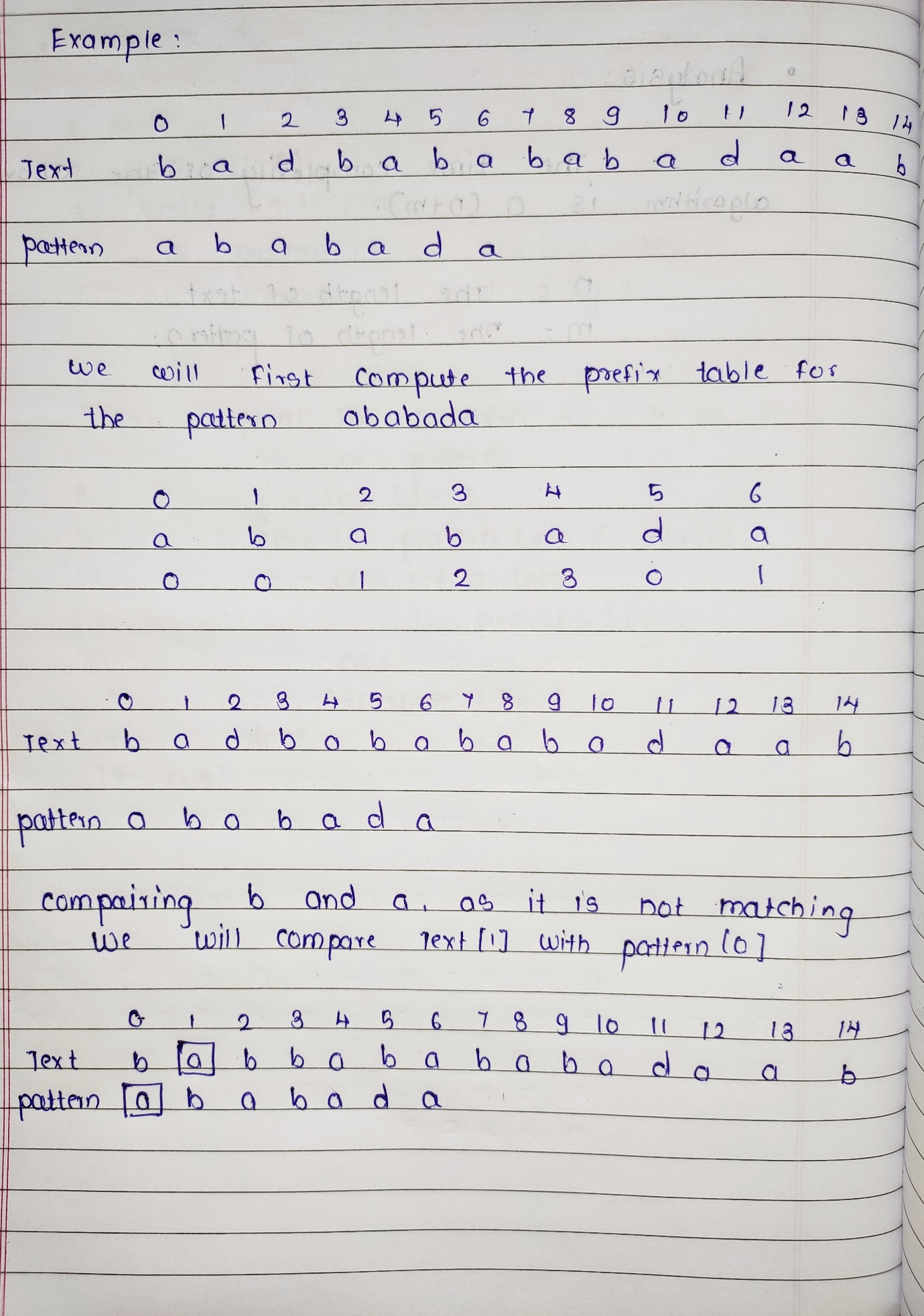


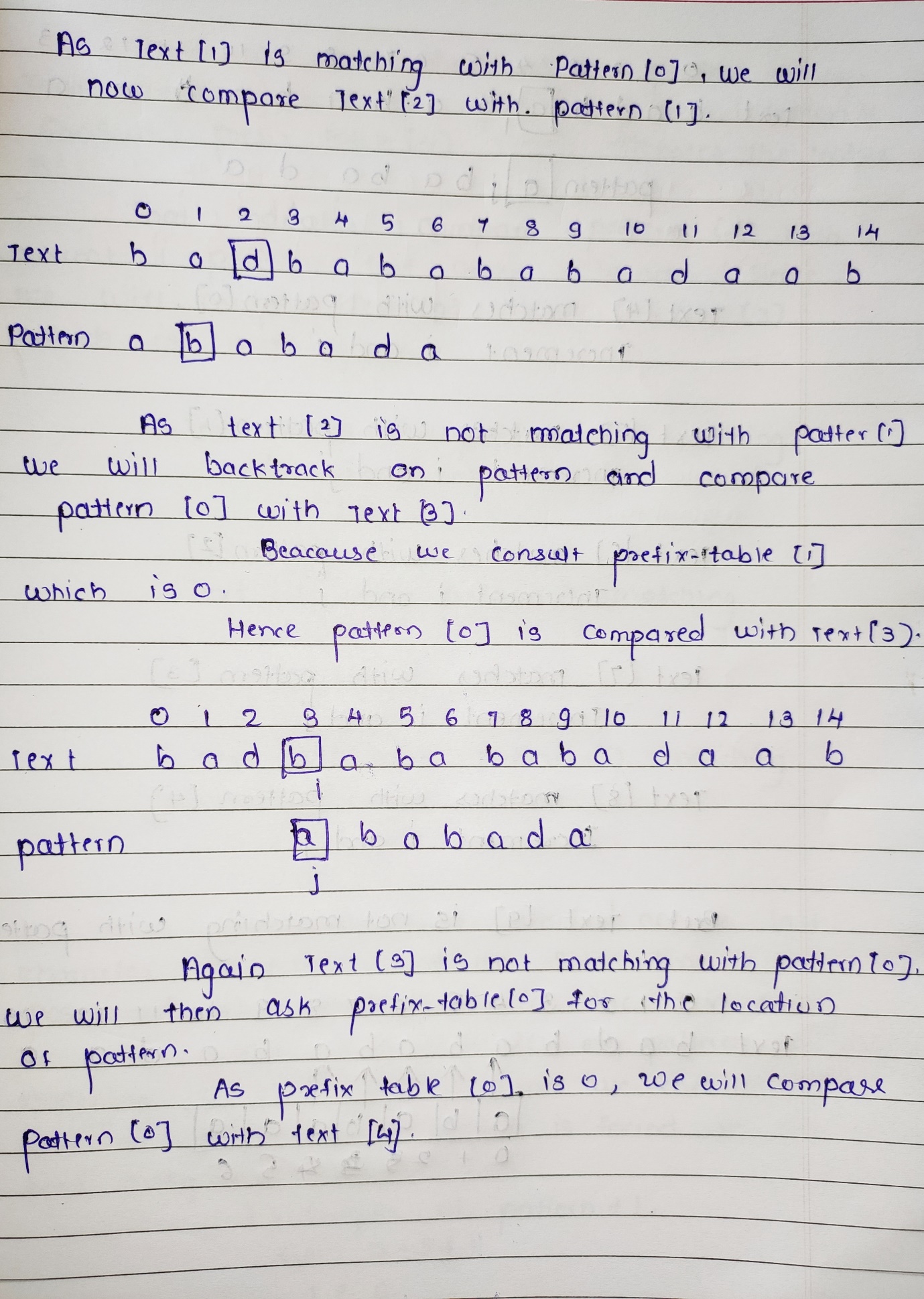


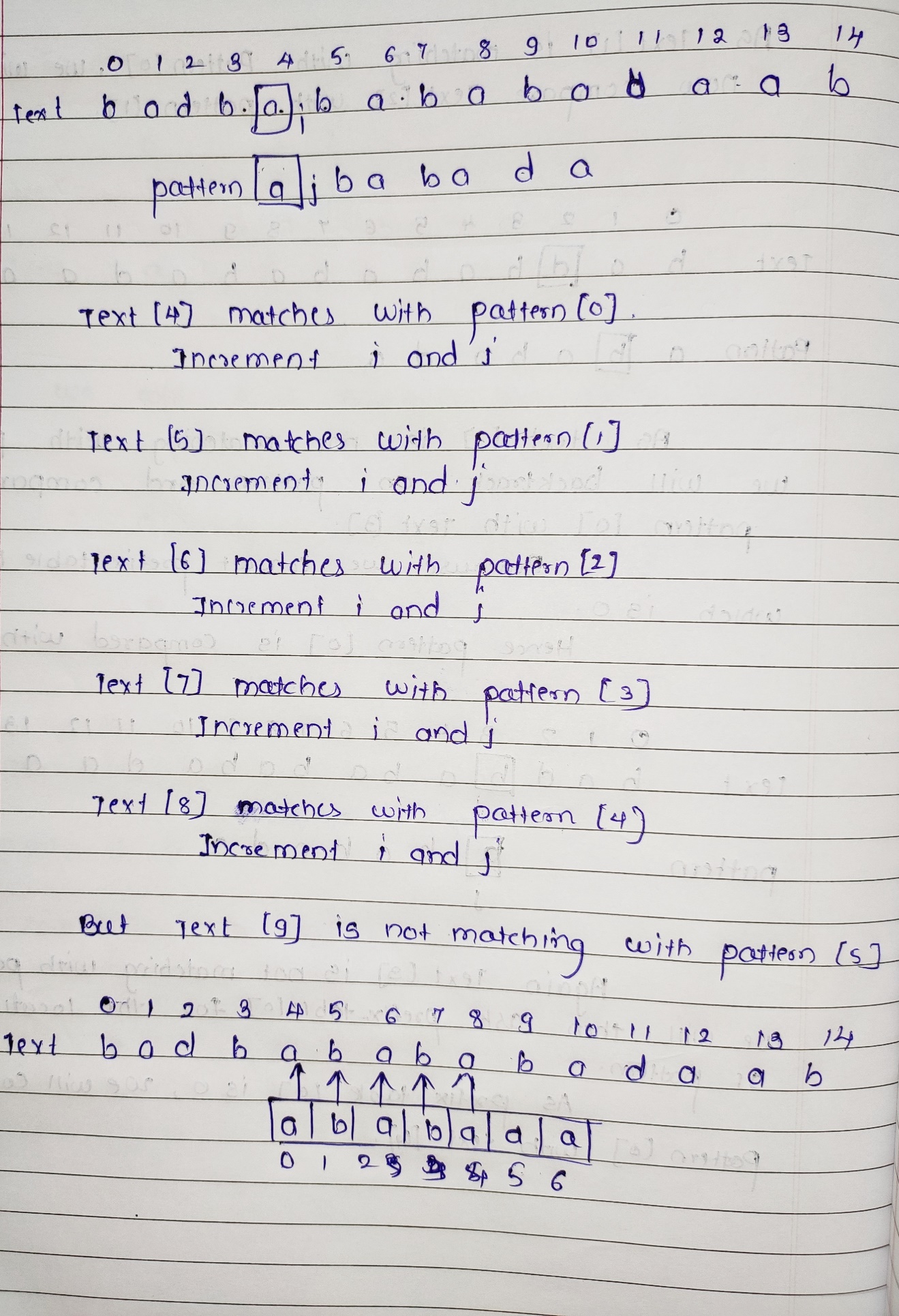
**Analysis**:

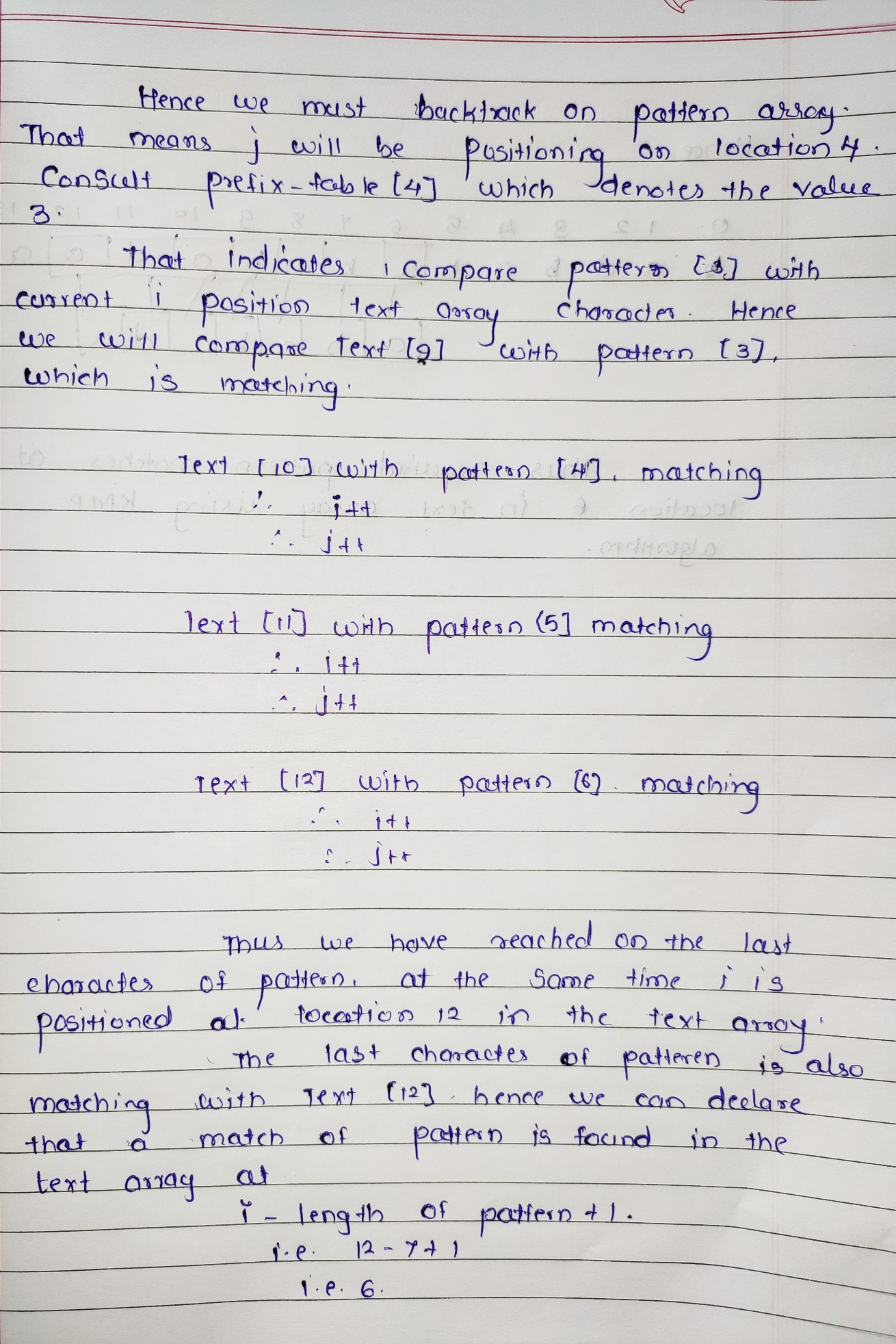


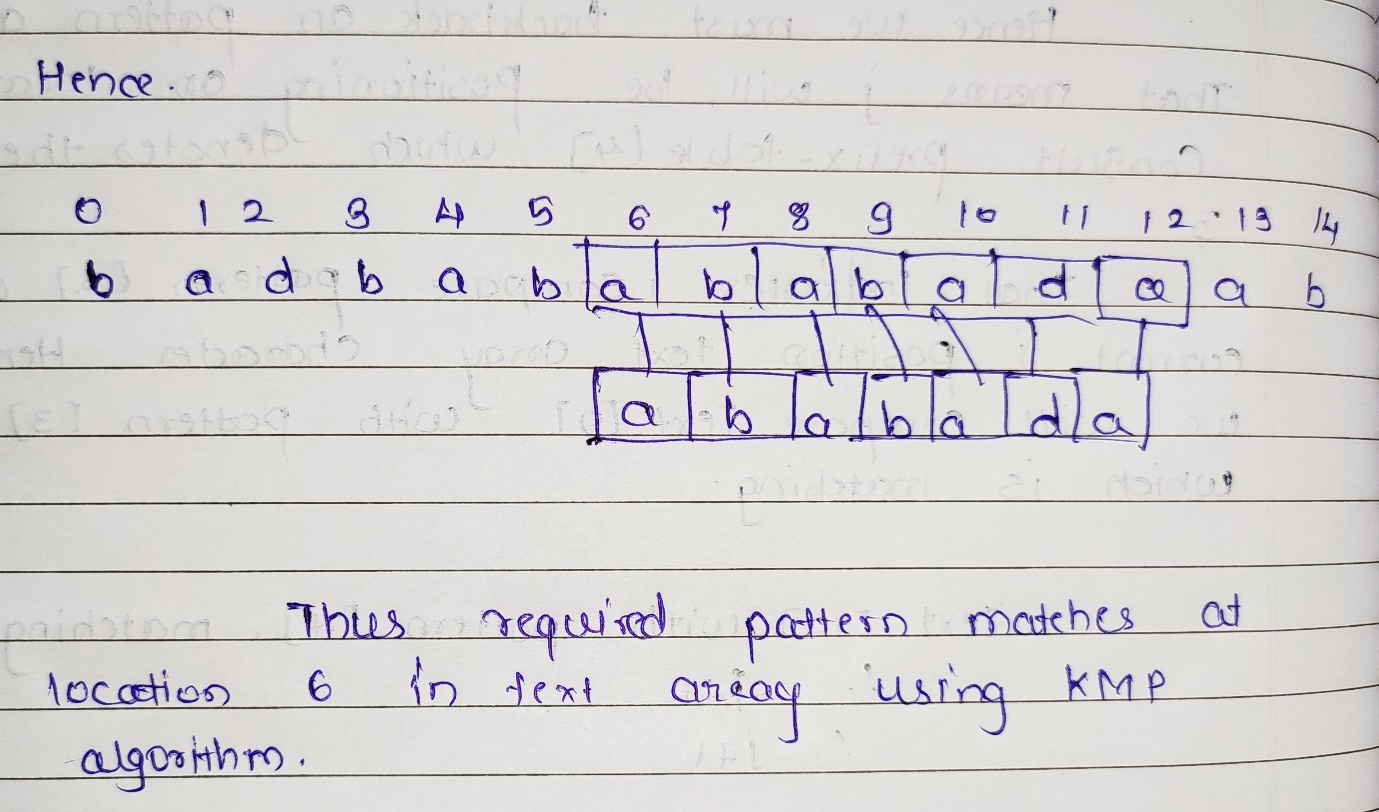
**Example**:











**Code**:

#include<stdio.h>

#include<stdlib.h>

#include<string.h>

char str[100],pattern[100];

int lps[100], locationArray[20]={[0 ... 19]=-1},ind;

int j=0,i=0,l1,l2;

void prefixtable()

{

int len = 0, i;

lps[0] = 0;

i = 1;

while(i < l2)

{

if(pattern[i] == pattern[len])

{

len++;

lps[i] = len;

i++;

}

else

{

if( len != 0 )

{

len = lps[len-1];

}

else

{

lps[i] = 0;

i++;

}

}

}

printf("\n\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*");

printf("\nLongest Prefix Sufix");

printf("\n");

printf("\t-------------------------------------------\n");

printf("\tPattern:|");

for(i=0;i<l2;i++)

{

printf(" %c |",pattern[i]);

}

printf("\n");

printf("\t-------------------------------------------\n");

printf("\tLPS: |");

for(i=0;i<l2;i++)

{

printf(" %d |",lps[i]);

}

printf("\n");

printf("\t-------------------------------------------\n");

}

void KMP()

{

int j=0,i=0;

prefixtable();

printf("\n");

while(i<l1)

{

if(pattern[j] == str[i])

{

j++;

i++;

}

if(j==l2)

{

locationArray[ind] = i-j;

ind++;

j = lps[j-1];

}

else if(pattern[j] != str[i])

{

if(j != 0)

{

j = lps[j-1];

}

else

{

i = i+1;

}

}

}

}

int main()

{

printf("\nEnter the String:");

scanf("%s",str);

printf("\nEnter the Pattern:");

scanf("%s",pattern);

l1=strlen(str);

l2=strlen(pattern);

KMP();

if(locationArray[0]==-1)

{

printf("\n\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");

printf("\tPattern not found.\n");

printf("\n\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");

}

else{

for(int i = 0; i<ind; i++) {

printf("\n\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");

printf( "\tPattern found at index: %d \n",locationArray[i] );

printf("\n\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");

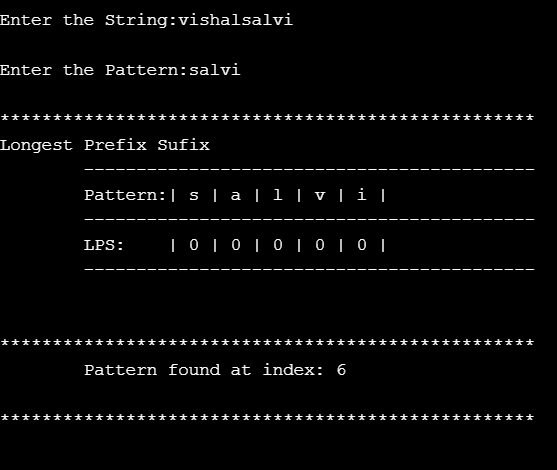
}

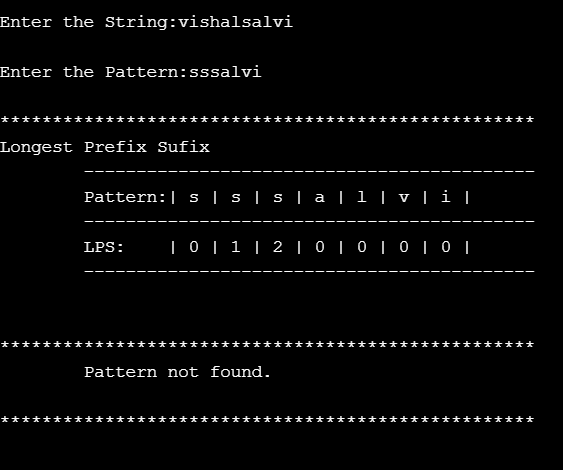
return 0;

}

}

**Output**:





**Conclusion:**

Thus, the basic idea behind this algorithm is to build a prefix array. This prefix array is built by using the prefix and suffix information of pattern. The overlapping prefix and suffix are used in LMP algorithm. The KMP algorithm achieves the efficiency of O(m+n) which is optimal in worst case.