Subject:
Date

-1

7 14 8 5 2 12

Insertion: 7 2 14 8 5 12 ①

→ 7 2 5 14 8 12 ②

→ 7 2 5 8 14 12 ③

→ 7 2 5 8 12 14 ④

الف) هرچه تعداد نابه‌جایی در آرایه T بیشتر باشد زمان اجرای الگوریتم بیشتر خواهد

ب) $T = a_0 \quad a_1 \quad a_2 \quad .. \quad a_n$

اگر بین a_i و a_j نابه‌جایی وجود داشته باشد به صورتی که $a_i > a_j$ ، $i < j$ پس
a_j باید $|j - i|$ بار به سمت چپ حرکت کند پس به تعداد $j - i$
تا نابه‌جایی داریم

-2

```
x = 0;                                              C1  n
for ( i=1 ; i <= n ; i++ ) {                        C2  n*n
    for ( j = 1 ; j <= n ; j++ )   x++;             C3  1
    j = 1 ;                                         C4  n/2
    while ( j < n ) {                               C5  2
    x++ ;   j = j * 2 ;
    }
}
```

5 n فرد → $[\frac{n}{2}]+1$

n زوج → $\frac{n}{2}$

$C_1 . n (C_2 . n + 1 + \frac{n}{2} + 2)$

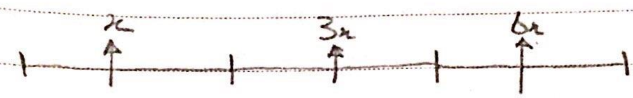
$= n (\frac{4}{5} n + 3) \Rightarrow T(n) = an^2 + bn$

Subject:
Date

3 -

O best case: 1 index = 0

Ω worst case: n index = n-1



$x + 3x + 6x = 1 \rightarrow x = \frac{1}{10}$

به احتمال 60% در $\frac{1}{3}$ سوم قرار دارد و احتمال قرار گرفتن آن در $\frac{1}{3}$ سوم هماش

وسط آن بازه است پس در حالت average انتظار می‌رود در $\frac{1}{3}$ سوم

آن بازه باشد یعنی $\frac{2}{3}n <$ index $a < n$ و از آنجایی که احتمال حضورش

در بازه (a, b) مساوی وسط بازه است پس انتظار می‌رود این تابع به

اندازه $\frac{5}{6}n$ باید حرکت کند پس:

Θ averag case: $\frac{5}{6}n$ یا $[\frac{2}{3}n, n]$

```c
#include <stdio.h>

#define SIZE 10

void swap(int *xp, int *yp)
{
    int temp = *xp;
    *xp = *yp;
    *yp = temp;
}

// A function to implement bubble sort
void bubbleSort(int arr[], int n)
{
   int i, j;
   for (i = 0; i < n-1; i++)

       // Last i elements are already in place
       for (j = 0; j < n-i-1; j++)
           if (arr[j] > arr[j+1])
              swap(&arr[j], &arr[j+1]);
}

int backwardLinearSearch(int *a, int b, int array[])
{
    for (int i = SIZE - 1; i >= 0; i--)
    {
        if(array[i] == b && &array[i] != a) return i;
        click++;
    }

    return -1;
}
```

```c
int main()
{
    int k = 10, array[SIZE] = {1, 8, 15, 5, 12, 7, 14, 25, 5};

    for (int i = 0; i < SIZE; i++)
    {
        // using pointer so the program dont use the same element.
        int *a = &array[i], b = k - *a, index;
        index = backwardLinearSearch(a, b, array);
        if (index != -1)
        {
            printf("%d + %d = %d\n",*a, array[index],k);
            time_complexity();
            return 0;
        }
    }

    printf("no match found!\n");
    return 0;
}
```

این کدی است که من برای سوال 4 در زبان سی استفاده کرده ام.

با توجه به این کد زمان اجرای برنامه در ارایه سورت شده و نشده تفاوت خوبی خواهد داشت!

توضیح ساده بخواهم بدهم الگوریتم در حالت كلي وقتي به يك عدد ميرسد ميايد و مكملش نسبت به كا را حساب ميكند و از انتهاي ارايه دنبالش ميگردد.

اگر ارايه سورت شده باشد وقتي به دنبال عدد مكمل ميگردد خيلي سريا تر به آن ميرسد چون عدد مكمل حتما از خود عدد بزرگ تر خواهد بود(در غير اين صورت در پيمايش هاي قبلي پيدا ميشد)، و چون ارايه سورت شده وقتي از انتها به ابتدا بياييد خيلي سريع تر پيدايش ميكند.

ولي اگر ارايه سورت نشده باشد ممكن است عدد مكملش قبل از خود عدد باشد و رسيدن به ان بيشتر طول ميكشد.

با همين مثالي كه در خود كد گذاشته ام اين موضوع به وضوح ديده ميشد

پس ارايه سورت شده پيچيدگي زماني كوتاه تري و اجراي كوتاه تري خواهد داشت.

Algorithm rate of growth = O(n^2)

a) $\log(n!) = \Theta(\log(n^n))$

$n! = n \times (n-1) \times (n-2) \times \dots \times 1$
$n^n = n \times n \times n \times \dots \times n$
$\Big\} \; n! \leq n^n$

$\Rightarrow \lg n! \leq \lg n^n \quad (\mathrm{I})$

$\exists c \in Q, \; \forall n \in N : \quad cn^n \leq n! \Rightarrow \lg cn^n \leq \lg n! \quad (\mathrm{II})$

$(\mathrm{I}), (\mathrm{II}) \longrightarrow \log n! = \Theta(\log(n^n))$

b) $n^{\frac{1}{\lg n}} = \Theta(1)$

$n^{\frac{1}{\lg n}} = n^{\lg_n^{10}} = 10 \Rightarrow n^{\frac{1}{\lg n}} = \Theta(1)$

c) $n! = \omega(2^n)$ $\qquad n! \approx A(n) = \left(\frac{n}{e}\right)^n \sqrt{2\pi n}$

$n! \sim \left(\frac{n}{e}\right)^n \sqrt{2\pi n} = n^{n+\frac{1}{2}} \times e^{-n} \sqrt{2\pi} > 2^n$

$n + \frac{1}{2} > n \rightarrow n^{n+\frac{1}{2}} > 2^n \rightarrow \left(\frac{n}{e}\right)^n \sqrt{2\pi n} > 2^n$
$n > 2$

$\Rightarrow \boxed{n! = \omega(2^n)}$

d) $n! = o(n^n)$

$n! = (\frac{n}{e})^n \sqrt{2\pi n}$

$\lim_{n \to \infty} \frac{\sqrt{2\pi n}}{e^n} = 0$

$n^n \times \frac{\sqrt{2\pi n}}{e^n} < n^n$

$\boxed{\Rightarrow n! = o(n^n)}$

(6

a) ✗ $g(n) = \omega(f(n))$

b) ✗ $y = x^2, \ y = x$

c) ✗ $f(n) = \lg(x), \ g(n) = \lg(x^2) \Rightarrow n < 10 \Rightarrow f(n) < 1$

d) ✓

e) ✗ $f(n) = -n \xrightarrow{f(n^2)} f(x^2) = -x^2$

f) ✓

g) ✓

h) ✓

Subject:
Date

is A $\square$ B

A	B	O	o	Ω	ω	Θ
n^2	n^3	✓	✓	✗	✗	✗
$\lg^k n$	n^{ε}	✗	✗	✗	✓	✓
n^k	c^n	✓	✓	✗	✗	✗
2^n	$2^{n/2}$	✗	✗	✓	✓	✓
$n^{\lg c}$	$c^{\log n}$	✓	✗	✓	✗	✓
$4^{\lg n}$	n^2	✓	✓	✗	✗	✗
$n!$	$n \cdot 2^n$	✓	✓	✗	✗	✗
$\sqrt{2}^{\lg n}$	$2^{\sqrt{2 \cdot \lg n}}$	✗	✗	✓	✓	✗
$(\lg(n))!$	2^{2n}	✓	✓	✗	✗	✗
$n^{\lg(\lg(n))}$	$(\lg(n))^{\lg n}$	✓	✗	✓	✗	✓

PAPCO