EXPERIMENT 10

20CP209P – Design and Analysis of Algorithm Lab

Aim:

To design and solve given problems using different algorithmic approaches and analyze their complexity.

1. Your friends are starting a security company that needs to obtain licenses for n different pieces of cryptographic software. Due to regulations, they can only obtain these licenses at the rate of at most one per month. Each license is currently selling for a price of $100. However, they are all becoming more expensive according to exponential growth curves: in particular, the cost of license j increases by a factor of r each month, where is a given parameter. This means that if license is $j > 1$ r j j purchased t months from now, it will cost $100r$. We will assume that all the price j t growth rates are distinct; that is, r for licenses (even though they start at the $i \neq r$ j $i \neq$ j sameprice of $100). The question is: Given that the company can only buy at most one license a month, in which order should it buy the licenses so that the total amount of money it spends is as small as possible? Give an algorithm that takes the n rates of price growth r , and 1 , r 2 , . . . , r n computes an order in which to buy the licenses so that the total amount ofmoney spent is minimized. The running time of your algorithm should be polynomial in n.
2. Suppose you are given an array A with n entries, with each entry holding a distinct number. You are told that the sequence of values A[1], A[2], . . . , A[n] is unimodal. That is, for some index p between 1 and n, the values in the array entries increase up to position p in A and then decrease the remainder of the way until position n. (So if you were to draw a plot with the array position j on the x-axis and the value of the entry $A[j]$ on the y-axis, the plotted points would rise until x-value p, where they'd achieve their maximum value, and then fall from there on). You'd like to find the "peak entry" p without having to read the entire array - in fact, by reading as few entries of A as possible. Show how to find the entry p by reading at most $O(log\ n)$ entries of A.

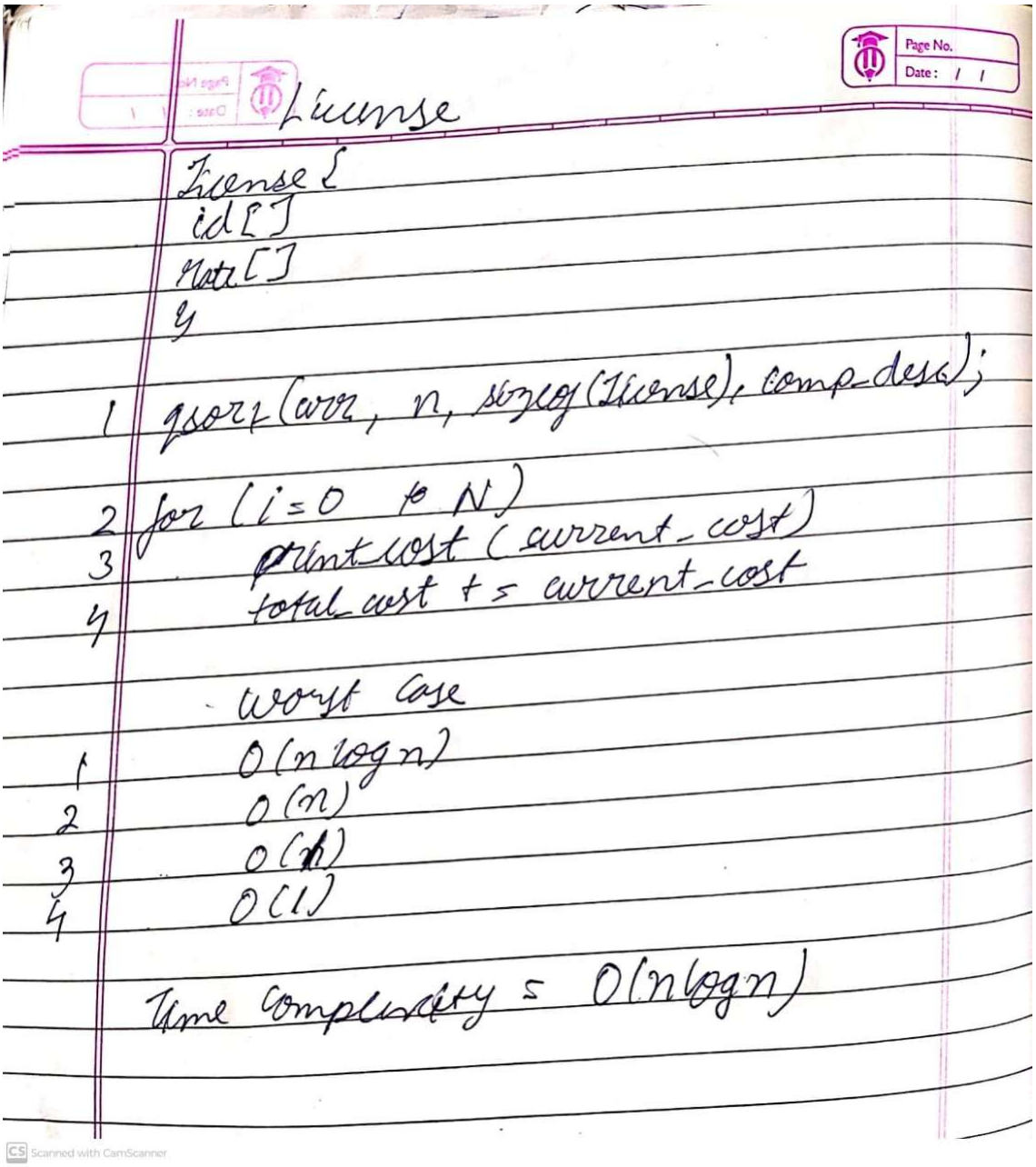
Code:

A (License):

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>

#define fr(i, a, b) for (int i = a; i < b; i++)

typedef struct License {
   int id;
   int cost;
} License;

int comp_desc(const void *a, const void *b);

int main(void)
{
   int r[] = {7, 8 , 9, 2, 3, 4, 5, 10, 3, 11};
   int n = sizeof(r) / sizeof(r[0]);
   License arr[9];
   fr (i, 0, n)
   {
      arr[i].id = i;
      arr[i].cost = r[i];
   }

   qsort(arr, n, sizeof(License), comp_desc);
   int ans[n];
   fr (i, 0, n)
   {
      ans[i] = arr[i].id;
   }
   double final_cost = 0;
   fr (i, 0, n)
   {
      int t = i;
      int rate = arr[i].cost;
      printf("Month => %d, Job => %d\n", i + 1, ans[i]);
      // final_cost += (100 * rate * t);
      final_cost += (100 * pow(rate, t));
   }
   printf("Final Cost => %.2f\n", final_cost);
   return 0;
}

int comp_desc(const void *a, const void *b)
{
   return(((License *)b)->cost - ((License *)a)->cost);
```

}

Analysis:


License
License {
id[]
rate[]
}
1 qsort(arr, n, sizeof(License), comp_desc);
2 for (i=0 to N)
3 print cost (current_cost)
4 total_cost += current_cost
- Worst case
1 O(n log n)
2 O(n)
3 O(1)
4 O(1)
Time complexity = O(n log n)
Page No.
Date : / /
Scanned with CamScanner

Output:

```
PS B:\sem4\23bcp153_daa\lab10> gcc license.c -o license
PS B:\sem4\23bcp153_daa\lab10> ./license
  Month => 1, Job => 9
  Month => 2, Job => 7
  Month => 3, Job => 2
  Month => 4, Job => 1
  Month => 5, Job => 0
  Month => 6, Job => 6
  Month => 7, Job => 5
  Month => 8, Job => 8
  Month => 9, Job => 4
  Month => 10, Job => 3
  Final Cost => 1948600.00
PS B:\sem4\23bcp153_daa\lab10>
```

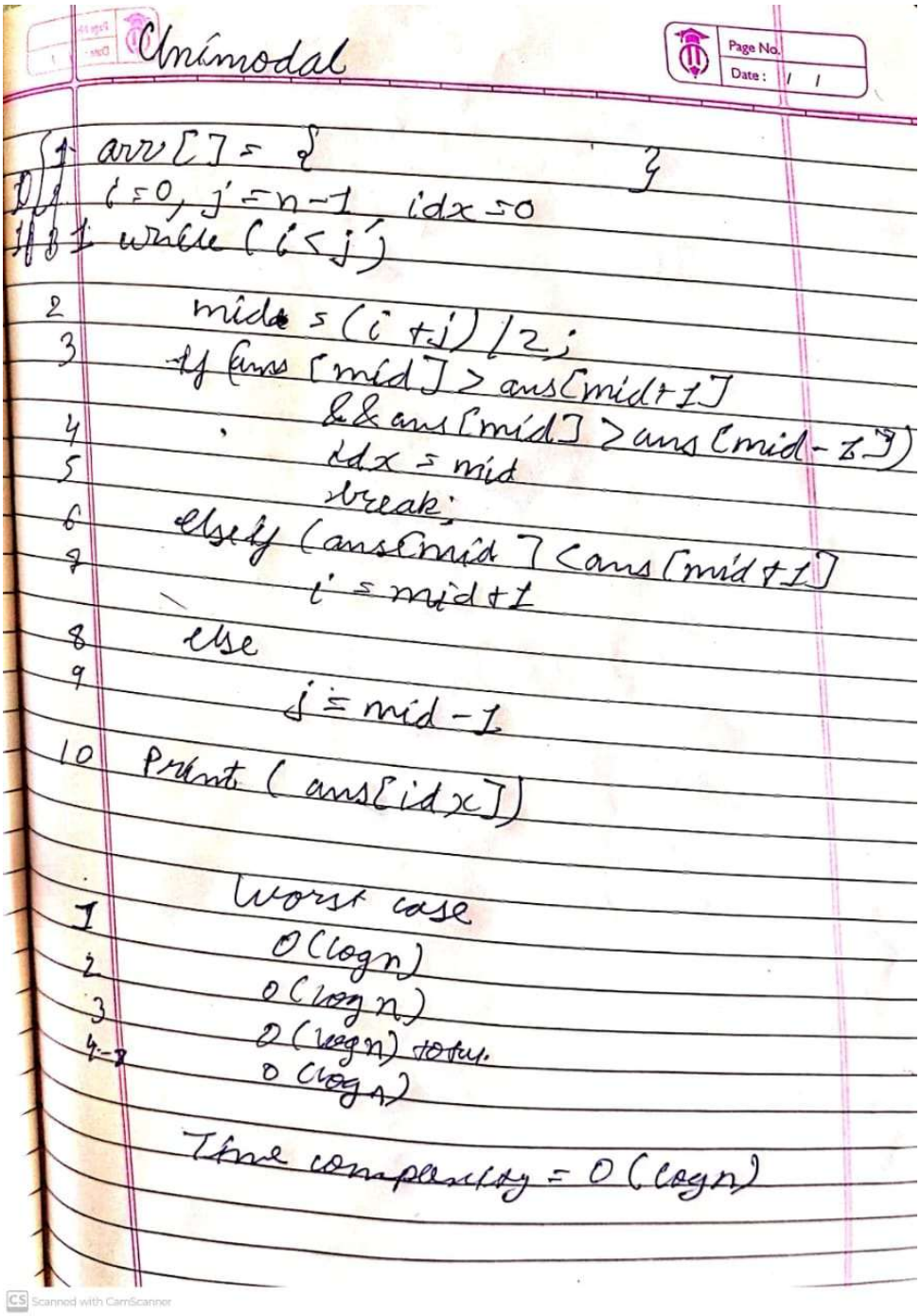
Code:

B (Unimodal):

```
#include <stdio.h>
#include <stdlib.h>

int main(void)
{
   int ans[] = {144, 206, 282, 576, 580, 1395, 1096, 1081, 694, 623};
   int n = sizeof(ans) / sizeof(ans[0]);

   int i = 0, j = n - 1;

   int idx = 0;
   int iter = 0;
   while (i < j)
   {
      iter++;
      int mid = (i + j) / 2;
      printf("%d\t", iter);
      if (ans[mid] > ans[mid + 1] && ans[mid] > ans[mid - 1])
      {
         idx = mid;
         break;
      }
      else if (ans[mid] < ans[mid + 1])
      {
         i = mid + 1;
      }
```

```
        else
        {
            j = mid - 1;
        }
    }

    printf("\nUnimodal peak found at index %d with value %d\n", idx, ans[idx]);
    return 0;
}
```

Analysis:



Output:

```
PS B:\sem4\23bcp153_daa\lab10> gcc unimodal.c -o unimodal
PS B:\sem4\23bcp153_daa\lab10> ./unimodal
1       2       3
Unimodal peak found at index 5 with value 1395
PS B:\sem4\23bcp153_daa\lab10>
```