ECE374 Fall2020

Homework5

Name: Zhang Yichi 3180111309

Oct. 25th 2020

1.2-1

When we try to book an apartment in airbnb, the app will provide plenty of information based on our commands. We can sort the result according to their prices or expected positions where sorting algorithms and graphs are involved or we can filter the result where tags and hash are involved.

1.2-2

$$\begin{gathered} 8n^2 < 64nlgn \\ n < 8lgn \\ 43 < 8lg43 \approx 43.4 \\ 44 > 8lg44 \approx 43.6 \\ \therefore n = 2, 3, \cdots, 43 \end{gathered}$$

1.2-3

$$\begin{gathered} 100n^2 < 2^n \\ n = 15, 22500 < 32768 \\ n = 14, 19600 < 16384 \\ \therefore n = 15 \end{gathered}$$

2.1-1

The input is $A = [31, 41, 59, 26, 41, 58]$.

In the 1*st* loop, it checks the numbers before the 2*nd* element 41 one by one in a loop and finds that 41 is just in its right place.
A becomes [31,41,59,26,41,58].

In the 2*st* loop, it checks the numbers before the 3*rd* element 59 one by one in a loop and finds that 59 is just in its right place.
A becomes [31,41,59,26,41,58].

In the 3*rd* loop, it checks the numbers before the 4*th* element 26 one by one in a loop, set them back if they are larger and finally insert 26 to the 1*st* place where the loop ends.
A becomes [26,31,41,59,41,58].

In the 4*th* loop, it checks the numbers before the 5*th* element 26 one by one in a loop, set them back if they are larger and finally insert 26 to the 4*th* place where the loop ends.
A becomes [26,31,41,41,59,58].

In the 5*rd* loop, it checks the numbers before the 6*th* element 26 one by one in a loop, set them back if they are larger and finally insert 26 to the 5*th* place where the loop ends.
A becomes [26,31,41,41,58,59].

2.1-2

New insertion sort which sort the list into nonincreasing order.

```
def insertion_sort(vec):
    for i in range(1,len(vec)):
        num = vec[i]
        j = i - 1
        while j >= 0 and vec[j] < num:
            vec[j+1] = vec[j]
            j -= 1
        vec[j+1] = num
```

2.1-3

```
def linear_search(vec,v):
    for i in range(len(vec)):
        if vec[i] == v:
            return i
    return -1
```

loop invariant: The first i-1 elements (vec[0],...,vec[i-1]) are not the value v.

Initialization: Return NIL if the list is empty. Or before the first loop iteration, there is no elements which shows that the loop invariant holds prior to the first iteration of the loop.

Maintenance: The first i-1 elements are not the value v. Thus, we will move on to check the i-th element.

Termination: If the i-th element is the value v, the function return the index i and terminate. Or if the loop terminates when $i > len(vec) = n$, which means $i = n + 1$ and first $i - 1 = n$ elements are not the value v, i.e. the value v is not in the list so return NIL.

2.2-1

$\Theta(\frac{n^3}{1000} - 100n^2 - 100n + 3) = \Theta(n^3)$

2.2-2

```python
def selection_sort(vec):
    for i in range(len(vec)-1):
        minn = i
        for j in range(i+1,len(vec)):
            if vec[j] < vec[minn]:
                minn = j
        vec[i],vec[minn] = vec[minn],vec[i]
```

loop invariant: The first (i-1) elements are the smallest (i-1) elements in the whole list.

It only needs to run for the first n-1 elements because if the first n-1 smallest elements are sorted, the rest one must be the largest and be put at the end of the list.

best-case: $\Theta(n + (n - 1) + \cdots + 1) = \Theta(\frac{n^2+n}{2}) = \Theta(n^2)$

worst-case: $\Theta(n + (n - 1) + \cdots + 1) = \Theta(\frac{n^2+n}{2}) = \Theta(n^2)$

2.3-1

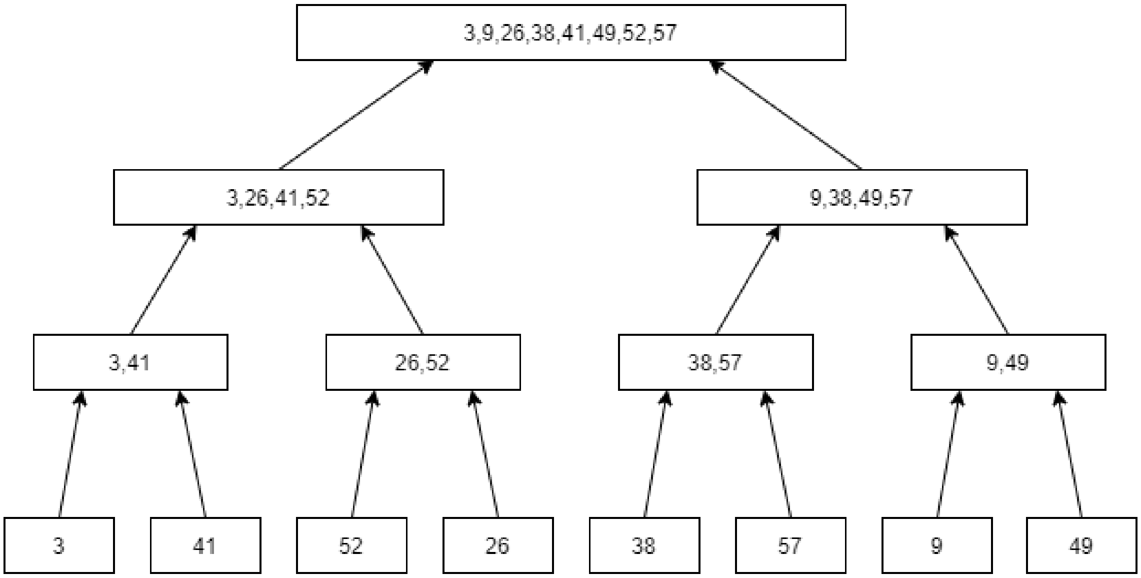

Figure 1: merge sort

2.3-2

```python
def merge(vec,mid):
    i = 0
    j = mid + 1
    n = len(vec)
    new_vec = []
    while (i <= mid and j < n):
        if vec[i] > vec[j]:
            new_vec.append(vec[j])
            j += 1
        else:
            new_vec.append(vec[i])
            i += 1
    if (i > mid):
        new_vec.extend(vec[j:])
    else:
        new_vec.extend(vec[i:mid+1])
    vec = new_vec
```

1-1

	1 second	1 minute	1 hour	1 day	1 month	1 year	1 century
lgn	2^{1e6}	2^{6e7}	$2^{3.6e9}$	$2^{8.64e10}$	$2^{2.592e12}$	$2^{3.1536e13}$	$2^{3.1536e15}$
$\sqrt{n}$	1e12	3.6e15	1.296e19	7.46496e21	6.718464e24	9.94519296e26	9.94519296e30
n	1e6	6e7	3.6e9	8.64e10	2.592e12	3.1536e13	3.1536e15
nlgn	62746	2.80142e6	1.33378e8	2.75515e9	7.18709e10	7.97634e11	6.8611e13
n^2	1000	7745	60000	293928	1609968	5615692	56156922
n^3	99	391	1532	4420	13736	31593	146645
2^n	19	25	31	36	41	44	51
n!	9	11	12	13	15	16	17