Assignment 1 of CISC 3000

ZHANG HUAKANG

March 4, 2022

1

I use a Hasp Map to count each number in A and B, and the number that the count is 1 is what we want find.

Algorithm 1:

```
H is a hash map.
for i ∈ A do
    if i ∈ H then
        H_i+=1
    end
    else
        H_i=1
    end
end
for i ∈ B do
    if i ∈ H then
        H_i+=1
    end
    else
        H_i=1
    end
end
for i ∈ H do
    if H_i = 1 then
        Output: i ∈ (A ∪ B) \ (A ∩ B)
    end
end
```

This algorithm is the count of each number in A or B is 1. We will prove that if the count of number is 1, this number must in $(A \cup B) \setminus (A \cap B)$

Proof. Since that elements in A have different values and elements in B also have different values, each element E in A or B will only show one time in its array, denoted as $No_A(E) = 1$ or $No_B(E) = 1$. If we put the elements in two array together into array C which allows duplicate, the number of each elements $No_C(E)$ will have two case. For element $E \in A \cup B$,

Case 1 If $E \in A$ and $E \notin B$, then $No_C(E) = 1$.

Case 2 If $E \notin A$ and $E \in B$, then $No_C(E) = 1$.

Case 3 If $E \in A$ and $E \in B$, then $No_C(E) = 2$.

So, if $No_C(E) = 1$, then ($E \in A$ and $E \notin B$) or ($E \notin A$ and $E \in B$), *i.e.*, if $E \in (A \cup B) \setminus (A \cap B)$ $\square$

Complexity The hash operation complexity is $O(1)$

$$
\begin{aligned}
T(|A| + |B|) =& |A| \times O(1) + |B| \times O(1) + |(A \cup B) \setminus (A \cap B)| \times O(1) \\
=& |A| + |B| + |(A \cup B) \setminus (A \cap B)| \\
<& |A| + |B| + |A| + |B| \\
=& 2(|A| + |B|) \\
=& O(|A| + |B|)
\end{aligned} \tag{1}
$$

2

a

Algorithm 2:

$p_L = 1$, $p_R = 1$, $count = 0$.
while $p_L \leq |L|$ *or* $p_R \leq |R|$ **do**
  if $L_{p_L} > R_{p_R}$ **then**
    $count += 1$
    $p_R += 1$
  end
  else
    $p_L += 1$
  end
end
Output: *count*

Complexity We have a pointer for each array. **For each while-loop, there must be only one poniter can move.** They both start from the begining of the array, and stop when both of them reach to the end of the array. p_L moves $|L|$ times, and p_R moves $|R|$ times. For the whole while-loop, it will be executed

$|L| + |R|$ times. We know that $count \in [0, |L| + |R|]$

$$
\begin{aligned}
T(|L| + |R|) =& 2 \times count + (|L| + |R| - count) \\
=& count + |L| + |R| \\
\leq& |L| + |R| + |L| + |R| \\
=& 2(|L| + |R|) \\
=& O(|L| + |R|)
\end{aligned}
\tag{2}
$$

b

```
function merge(A : array, p : int, q : int, r : int)
    n_1 = q - p + 1
    n_2 = r - q
    let L[1..n_1 + 1] and R[1..n_2 + 1]
    for i = 1 to n_1 do
        L[i] = A[p + i - 1]
    end
    for i = 1 to n_2 do
        R[i] = A[q + j]
    end
    L[n_1 + 1] = R[n_2 + 1] = ∞
    i = j = 1
    for k = p to r do
        if L[i] ≤ R[j] then
            A[k] = L[i]
            i+ = 1
        end
        else
            A[k] = R[i]
            j+ = 1
        end
    end
end
function merge-sort(A : array, p : int, r : int)
    if p < r then
        q = ⌊(p + r)/2⌋
        merge-sort(A,p,q)
        merge-sort(A,q+1,r)
        // Count the inversion between A[p, q] and A[q + 1, r],
         O(q − p + 1)
        count-inversion(A,p,q,r)
        merge(A,p,q,r)
    end
end
```

Complexity It is easy to find that

$$\begin{aligned} T_{merge}(p,r) =& r-p+1 \\ =& O(r-p) \end{aligned} \tag{3}$$

$$\begin{aligned} T(n) =& 2\times T(\frac{n}{2}) + T_{merge}(1,n) + cn \\ =& 2\times T(\frac{n}{2}) + c'n \\ =& 2\log n \times c'n + c'n \\ =& O(n\log n) \end{aligned} \tag{4}$$

3

Proof. $T(1), T(2), T(3) \le c = O(1)$ and

$$T(n) \le T(\frac{n}{4}) + T(\frac{3}{4}) + cn \tag{5}$$

when $n \ge 4$. Thus,

$$\begin{aligned} T(4) \le& T(1) + T(3) + cn \\ =& 2c + 4c \\ =& 6c \\ \le& \frac{6c}{4\log 4} 4\log 4 \\ =& O(n\log n) \end{aligned} \tag{6}$$

Suppose that $\forall n \in [4, k-1], T(n) = O(n\log n)$, we have

$$\begin{aligned} T(k) \le& T(\frac{k}{4}) + T(\frac{3k}{4}) + ck \\ \le& c_1\frac{k}{4}\log\frac{k}{4} + c_2\frac{3k}{4}\log\frac{3k}{4} + ck \\ =& \frac{c_1k}{4}\log\frac{k}{4} + \frac{c_2 3}{4}(\log\frac{k}{4} + \log 3) + ck \\ =& \frac{c_1k + 3c_2k}{4}\log\frac{k}{4} + \frac{3\log 3c_3}{4} + ck \\ =& O(k\log k) \end{aligned} \tag{7}$$

Thus, $\forall n \ge 4$, $T(n) = O(n\log n)$ $\square$

4

```python
def countSort(arr:list, n:int, exp:int)->None:
    output = [0] * n
    count = [0] * n
    for i in range(n):
        count[i] = 0
    for i in range(n):
        count[ (arr[i] // exp) % n ] += 1
    for i in range(1, n):
        count[i] += count[i - 1]
    for i in range(n - 1, -1, -1):
        output[count[ (arr[i] // exp) % n] - 1] = arr[i]
        count[(arr[i] // exp) % n] -= 1
    for i in range(n):
        arr[i] = output[i]
if __name__ =="__main__":
    arr = [33, 1, 22, 40, 12, 45, 32]
    n = len(arr)
    countSort(arr, n, n)
    print(arr)
```

Complexity

$$\begin{aligned} T(n) =& n + n + n + n \\ =& 4n \\ =& O(n) \end{aligned} \tag{8}$$

5

a

```
Input: A
MergeSort(A)
max_time = −1
t_n = A_0
t = 1
for i = 1 to lenght(A) − 1 do
    if A_i == t_n then
        t+ = 1
    end
    else
        if t ≥ max_time then
            max_time = t
        end
        t_n = A_i
        t = 1
    end
end
if max_time > length(A) then
    Output :Yes
end
else
    Output :No
end
```

Correctness After sort this array, the numbers that have same value will be together. We count the number of each number in array and record the maximum value of it. If the maximum value is greater that the half of the array length, then we can say that more than half of the numbers have the same value.

Complexity

$$\begin{aligned} T(n) =& T_{mergesort}(n) + (c_1 + 2 \times (n - c_2) + c_3) \\ =& O(n \log n) + O(n) \\ =& O(n \log n) \end{aligned} \tag{9}$$

b

```
a=[2,2,2,1]
d={}
for i in a:
    if i in d:
        d[i]+=1
    else:
        d[i]=1

flag=False
for i in d:
    if d[i]>len(a)/2:
        flag=False
        print("yes")
    break
if not flag:
    print("No")
```

Correctness We count the number of each number in array and record it in a hash map. If there is a value that is greater that the half of the array length, then we can say that more than half of the numbers have the same value.

Complexity

$$
\begin{aligned}
T(n) =& n \times O(1) + n \\
=& 2n \\
=& O(n)
\end{aligned} \tag{10}
$$