(۱)

```
def common_elements_count(a,b):
    a sort --> a
    sort --> b
    i,j = 0,0
    common = 0
    while (i < size.a and j < size.b)
        if a[i] == b[j] --> common++ , i++, j
        else if a[i] < b[j] --> i++
        else: --> j++
    return common
```

اول دو مجموعه را باید sort کنیم. سپس با دو pointer بین element های آن دو مجموعه می رویم. همیشه pointer که به element کمتر اشاره می کند را پیش می بریم. حال اگر به هردو به دو مقدار یکسان اشاره کند، یک عضو مشترک داریم.

$O(n \log n + m \log m)$

n: سایز مجموعه اول

m: سایز مجموعه دوم

bucket Sort → به جای تقسیم به n زیر بازه تقسیم

به k زیر بازه $k < n$

A (1, n) تا n می‌گیره   B (0, k) تا n می‌گیره → B(0, k)

```
for (i = 1 to n)
    insert A(i) into list B(⌊kA(i)⌋)    → θ(n)
for (i = 0 to k-1)
    sort  ⇝   چون k<n  →
print
```

اگه بخواهیم از الگوریتم mergeSort استفاده کنیم به خاطر اینکه همه n تا در یک خانه باشند

$n \lg n$

و در بدترین حالت n^2 می‌شود

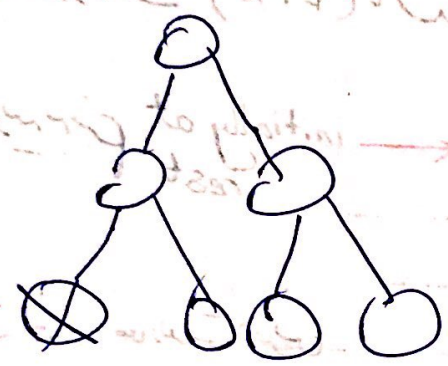
بستگی به k دارد و در هر صورت مرتبه الگوریتم $O(n^2)$ می‌باشد

3

درخت باینری - درهر استک، نام ومشخصات بیمار را می نویسیم

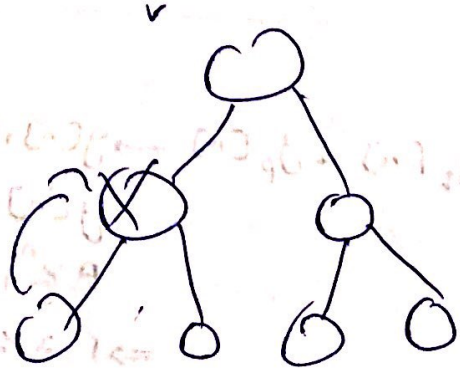
تا از تکراری شدن عنصر جلوگیری کنیم. حال با استفاده از حروف اول هریک از مشخصات درخت خود را مرتب می کنیم.

برای delete کردن $O(H)=O(\log n)$ می رویم به سراغ پایین ترین عنصر و چپ ترین عنصر در درخت را حذف می کنیم



یک حالت دیگر برای حذف کردن، حذف عنصری با دو فرزند است.

ابتدا به وسیله Query نیاز به جستجو داریم. درخت خود را پیدا کرده وحذف می کنیم سپس فرزند را به جای آن می گذاریم.



برای insert، درخت خود را به نحوی مرتب می کنیم که سمت چپ زن و سمت راست مرد باشد. سپس براساس یک عنصر آن (مثلا سن) آن ها را مرتب کرده و حالا برای insert طبق آن عنصر، insert می کنیم.