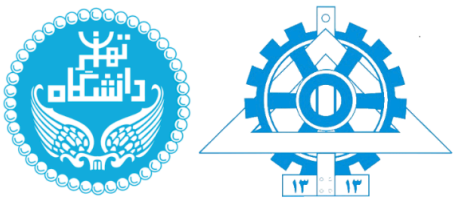
به نام خدا



دانشگاه تهران، دانشکده مهندسی برق و کامپیوتر
تحلیل و طراحی الگوریتم‌ها

تمرین کتبی اول
پاسخنامه
طراح: امیرحسین احمدی amirhmi1377@gmail.com

---

۱. ابتدا لیست رشته های داده شده را به دو بخش تقسیم می کنیم. حال همین کار را بر روی قسمت چپ و قسمت راست انجام می دهیم. این کار را تا جایی انجام می دهیم که تمام لیست ها، دارای حداکثر یک رشته باشند. در این مرحله شروع به به دست آوردن بزرگترین زیر رشته می کنیم، به این صورت که در هر مرحله بزرگترین زیر رشته دو لیست مجاور را به دست آورده، و سپس آن دو را به هم می چسبانیم و زیر رشته به دست آمده را به عنوان جواب این لیست ساخته شده در نظر می گیریم. در آخرین مرحله که از ادغام دو لیست رشته سمت چپ و راست جواب برابر زیر رشته مشترک آن دو می باشد.

۲. از تعریف درخت bst استفاده می کنیم. درخت bst درختی می باشد که مقدار هر گره از ماکسیموم مقدار گره های زیردرخت سمت چپ بزرگتر و از مینیموم مقدار گره های زیردرخت سمت راست کوچکتر باشد. حال در هر مرحله چک می کنیم که آیا مقدار گره از کوچکترین مقدار سمت راست کوچکتر و از بزرگترین مقدار سمت چپ بزرگتر می باشد یا خیر که برای این کار می دانیم سمت راست ترین گره در زیر در درخت سمت چپ بزرگترین مقدار را دارد و سمت چپ ترین گره در درخت سمت راست نیز کوچکترین مقدار را دارا می باشد. حال اگر این شروط را دارا باشد و زیردرخت سمت چپ و راست آن نیز شروط bst را داشته باشند، درخت bst می باشد.

```
int isBST(struct node* node)
{
  if (node == NULL)
    return(true);
  if (node->left!=NULL && maxValue(node->left) > node->data)
    return(false);
  if (node->right!=NULL && minValue(node->right) < node->data)
    return(false);
  if (!isBST(node->left) || !isBST(node->right))
    return(false);
  return(true);
}
```

الگوریتم گفته شده در بالا از نظر مرتبه زمانی بهینه نمی باشد چونکه ممکن است هر نود را چندین مرتبه پیمایش کند. یکی از روش های بهینه اسن است که هر نود مقدار کوچکترین و بزرگترین زیر درختش را برگرداند و هر نود اگر از بزرگترین مقدار زیر درخت سمت چپ بزرگتر باشد و از کوچکترین مقدار زیر درخت سمت راست، بزرگتر باشد، درخت داده شده bst می باشد و مرتبه زمانی آن O(n) می باشد چون تنها یک بار هر راس ملاقات می شود.

روش دوم : یک روش دیگر حل این مسئله، استفاده از پیمایش میان ترتیب می باشد به این صورت که در هر مرحله مقدار قبلی را نگه داریم و در صورتی که مقدار نود فعلی بزرگتر مساوی مقدار نود قبلی بود، در خت bst می باشد.

۳. در هر مرحله آرایه ورودی را به قسمت می شکنیم و تابع را روی ورودی جدید صدا می زنیم تا زمانی که به یک آرایه با یک عنصر برسیم. در یک آرایه با یک عنصر، همان عنصر جواب می باشد و برگردانده می شود. در هنگام مرج کردن دو زیر آرایه نیز چهار حالت پیش می آید که به صورت زیر عمل می کنیم:
یک) هیچ کدام از دو زیر آرایه عنصری ندارند که بیش از نصف تکرار شده باشد. پس آرایه حاصل از مرج شدن نیز چنین عنصری ندارد.
دو) زیر آرایه سمت چپ عنصری با تکرار بیشاز نصف دارد ولی سمت راست ندارد که با پیمایش روی تمام عناصر هر دو آرایه، اگر تعداد تکرار این عنصر از نصف بیشتر بود، به عنوان عنصر پرتکرار برگردانده می شود.
سه) زیر آرایه سمت راست چنین عنصری دارد و زیر آرایه سمت چپ ندارد که همانند قسمت دو عمل می کنیم.
چهار) هر دو زیر آرایه عنصر پر تکرار دارند که در این صورت تعداد تکرار هر دو عنصر را در آرایه حاصل از مرج شدن دو زیر آرایه به دست می آوریم و اگر یکی از آن ها تعداد تکرارش بیشتر از نصف آرایه حاصل از مرج شدن بود، آن را به عنوان پاسخ بر می گردانیم.

۴. در هر مرحله آرایه را به دو قسمت چپ (L) و راست (R) تقسیم می کنیم. هر قسمت در خروجی خود می بایست ۴ مقدار totalSum , maxSum, maxPrefix, maxSuffix را محاسبه و برگرداند. حال می دانیم که totalSum برابر L.totalSum + R.totalSum می باشد. حال برای مقدار maxPrefix اکر نخواهد که از وسط آرایه عبور کند، مقدارش برابر L.maxPrefix می باشد و در صورتی که این محدودیت نباشد برابر L.totalSum + R.maxPrefix می باشد. در نهایت maxSum را محاسبه می کنیم. این مقدار برابر ماکسیموم R.totalSum + L.maxSuffix و R.maxSuffix می باشد.
مرتبه زمانی : $T(n) = 2T(n/2) + d$ که برابر $O(n)$ می باشد.

```
function FMS-COMPARE(A,low,high)
        if low = high then
                return (A[low], A[low], A[low], A[low])
        else
                mid ← (low + high) / 2
                Left ← FMS-COMPARE(A, low, mid)
                Right ← FMS-COMPARE(A, mid + 1, high)
                return COMPARE(A,Left,Right)
        end if
end function

function COMPARE(A,L,R)
        totalSum ← L.totalSum + R.totalSum
        maxPrefix ← MAX(L.maxPrefix,L.totalSum + R.maxPrefix)
        maxSuffix ← MAX(R.maxSuffix, R.totalSum + L.maxSuffix)
        maxSum ← MAX(L.maxSum, R.maxSum,
        L.maxSuffix + R.maxPrefix)
        return (totalSum,maxSum,maxPrefix, maxSuffix)
end function
```

۵. یک آرایه را unimodal می نامیم هر گاه از ابتدای آرایه تا جایی اعداد به صورت صعودی باشند و از آنجا به بعد به صورت نزولی. حال ادعا می کنیم این تعییر جهت در چنین آرایه ای را می توان در O(lgn) بدست آورد. ابتدا عنصر میانی را نگاه می کنیم و اگر از عدد قبل بزرگتر. و از عدد بعد کوچکتر باشد در قسمت صعدی قرار دارد و می توانیم تغییر را در نیمه دوم آرایه که آن هم unimodal است جستحو کنیم، همین طور اگر از عدد قبل کوچکتر و از عدد بعد بزرگتر بود در قسمت نزولی قرار دارد و جواب در نیمه اول آرایه است. و اگر از عنصر بعد بزرگتر و از عنصر قبل هم بزرگتر باشد، عنصر مورد نظر می باشد.
به همین صورت مانند جست و جوی دودویی ادامه می دهیم تا این عنصر را پیدا کنیم. حال توجه می کنیم که این نقاط بر حسب مولفه x، unimodal هستند و نقطه با بیشترین x هم همان نقطه تغییر جهت است. پس. در O(lgn) می شود آن را پیدا کرد. سپس اگر از این نقطه تا انتهای نقطه ها به آرایه نگاه کنیم بر حسب مولفه y، unimodal است و عنصر با بیشترین y نقطه تفییر جهت است و آن را می توان در O(lgn) پیدا کرد.

۶. ابتدا عنصر میانه را پیدا می کنیم. حال روی آرایه حرکت می کنیم و وزن عناصری را که از میانه کوچک تر هستند جمع می کنیم و این مقدار را k می نامیم. اگر k از $1/2$ بزرگتر بود میانه وزن دار در نیمه اول آرایه قرار دارد یعنی کافیست عناصر کوچکتر از میانه را در یک آرایه دیگر برزیم و آخرین عنصری در این آرایه که جمع وزن ها تا آنجا از $1/2$ کوچکتر است را پیدا کنیم. و اگر از $1/2$ کوچکتر باشد این عنصر در نیمه دوم آرایه قرار دارد و باید در نیمه دوم به دنبال آخرین عنصری بگردیم که جمع وزن از میانه تا آنجا حداکثر $k - 1/2$ باشد. به همین صورت می توان بازگشتی عمل کرد و در هر مرحله میانه آرایه فعلی را پیدا کرد و این اعمال را انجام داد.
تحلیل زمان اجرا : $T(n) = T(n/2) + O(n)$ که برابر $O(n)$ می باشد.