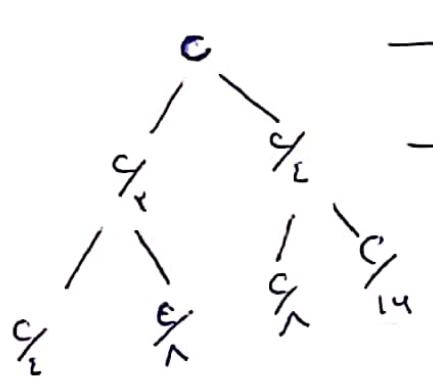
محمدمهدی تهاری - ۹۹۳۱۰۶۱

(۱)

(۹

$$T(n) = T(\tfrac{n}{2}) + T(\tfrac{n}{4}) + O(1)$$

حل: روش درخت بازگشت



$\rightarrow c$

$\rightarrow (\tfrac{3}{4})c$

$\rightarrow (\tfrac{9}{16})c = (\tfrac{3}{4})^2 c$

تعداد برگ در سطح k $= (\tfrac{3}{4})^k c$ $\Rightarrow$

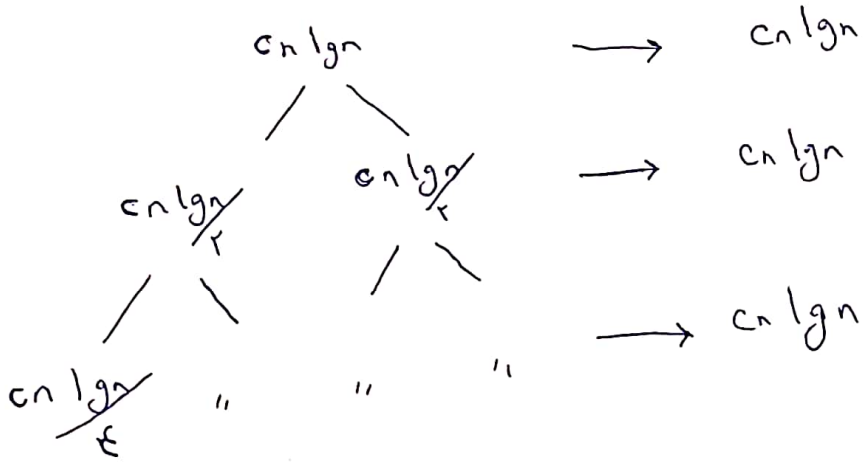
$$T(n) = \sum_{k=0}^{\infty} (\tfrac{3}{4})^k c = c \sum_{k=0}^{\infty} (\tfrac{3}{4})^k = c \frac{1\left(1 - (\tfrac{3}{4})^{\infty}\right)}{1 - \tfrac{3}{4}} = c \frac{1(1-0)}{1/4}$$

$$= 4c = O(1)$$

b)

$T(n) = 2T(n/2) + cn\lg n$

حل به روش درخت بازگشتی



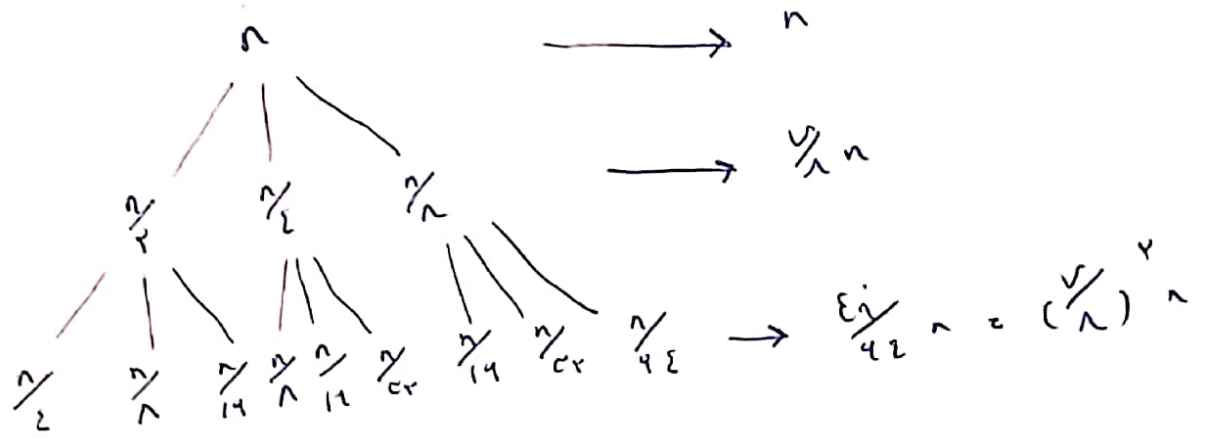
میزان پردازش هر سطح = $cn\lg n$

طبق دلیل آنکه در هر سطح نصف ورودی سطح قبل است و تعداد سطح ها $\lg n$ است پس داریم

$T(n) = \lg n \times cn\lg n = cn^2\lg n = O(n^2\lg n)$

(۲)

a) حل به روش درخت بازگشتی



تعداد برگ‌ها = $(\frac{7}{8})^k n \Rightarrow (\frac{7}{8})^2 n = \frac{49}{64} n \rightarrow 1$

$$T(n) = \sum_{k=0}^{\infty} (\tfrac{7}{8})^k n = n \sum_{k=0}^{\infty} (\tfrac{7}{8})^k = 8n = O(n)$$

$$\sum_{k=0}^{\infty} (\tfrac{7}{8})^k = \frac{a(1-q^{n+1})}{(1-q)} = \frac{1\left(1-(\frac{7}{8})^{\infty}\right)}{1-\frac{7}{8}} = \frac{1(1-0)}{\frac{1}{8}} = 8$$

b) فرض می‌کنیم $n = 2^m$ و در نتیجه $\sqrt{n} = 2^{m/2}$ ، در این صورت:

$$T(n) = \sqrt{n}\,T(\sqrt{n}) + n \rightarrow T(2^m) = 2^{m/2}T(2^{m/2}) + 2^m \xrightarrow{\times\, 2^{-m}}$$

$$\frac{T(2^m)}{2^m} = \frac{T(2^{m/2})}{2^{m/2}} + 1$$

فرض می‌کنیم $S(m) = \frac{T(2^m)}{2^m}$ پس داریم

$$S(m) = S(m/2) + 1$$

حال طبق قضیه اصلی با $a=1$ ، $b=2$ ، $f(n)=1$

داریم $S(m) = \Theta(\lg m)$

$$S(m) = \Theta(\lg m) \rightarrow \frac{T(2^m)}{2^m} = \Theta(\lg m) \Rightarrow T(2^m) = 2^m\Theta(\lg m) \rightarrow$$

$$T(n) = n\,\Theta(\lg\lg n) = \Theta(n \lg\lg n)$$

④ با استفاده از یک min-heap یا priority queue این کار را انجام می‌دهیم به شکلی که ابتدا با k تا عنصر اول آرایه یک min heap می‌سازیم. سپس برای اعضای بعدی آرایه به ترتیب اول کوچکترین عضو min heap را خارج کرده و آرایه جزوی را ترمیم و عضو بعدی آرایه را در min heap وارد می‌کنیم. این کار را برای همه اعضای لیست تا زمانی که min heap خالی شود. برای تحلیل زمانی می‌دانیم که insert هر عنصر در min heap از مرتبه $O(\log k)$ است و برای همه‌ی n عضو آرایه این کار را با یک تکرار لیست پس از مرتبه $O(n \log k)$ است.

(۴) نام این الگوریتم d-way quicksort است که در این الگوریتم به جای انتخاب ۱ pivot و تقسیم آرایه به دو بخش، $d-1$ pivot انتخاب کرده و آرایه را به d بخش تقسیم می‌کنیم.

روش انتخاب pivot همانند حالت عادی quicksort است. در هر مرحله partitioning عنصرهای کمتر از pivot اول در مرحله اول، عنصرهای بین pivot اول و دوم در مرحله دوم و به همین ترتیب تا آخر قرار می‌گیرند. سپس برای هر مرحله بازگشتی همین مراحل را برای هر مرحله تکرار می‌کنیم و در آخر مرحله‌های مرتب‌شده را با هم Concat می‌کنیم. برای تحلیل پیچیدگی زمانی به رابطه بازگشتی زیر می‌رسیم:

$$T(n) = dT(n/d) + O(n)$$

که $O(n)$ هزینه زمانی برای هر Partition در هر مرحله هست

و با توجه به master theory داریم

$$T(n) = O(n \lg n)$$

۵)

a) برای والد یک فرزند داریم : $parent(i) = \lfloor \frac{i}{d} \rfloor$

و برای k امین فرزند والد i داریم : $child(k,i) = d(i-1) + k + 1$

b) حداقل بازی heap به ارتفاع $\lfloor \log_d n \rfloor$ ، در d-ary Heap داریم $\lfloor \log_2 n \rfloor$

c) ابتدا سایز heap را 1 واحد افزایش می‌دهیم و عنصر جدید را در آخرین خانه آرایه قرار می‌دهیم.
سپس آن را مرحله به مرحله با Parent خود چک می‌کنیم و اگر از والد خود بزرگتر باشد آن را با آن جابجا می‌کنیم تا زمانی که به ریشه برسیم. شبه کد:

```
def insert(array, key):
    array.size += 1
    array[array.size-1] = key
    i = array.size -1
    while i >1 and array[i] > array[Parent(i)]:
        array[i], array[parent(i)] = array[parent(i)], array[i]
        i = parent(i)
```

محل جابجایی با parent