هر سوال را در محل در نظر گرفته شده پاسخ دهید. پاسخ های خارج از محل تصحیح نمیشوند. نام و شماره دانشجویی را روی **تمام برگه ها** بنویسید. شماره دانشجویی باید با **اعداد لاتین** نوشته شود. مهلت این تمرین شنبه ۱۶ آذر ماه است.

غزاله محمودی ـ سید صالح اعتمادی

۱. [۱۰] برای آرایه مقابل الگوریتم Merge sort را مرحله به مرحله اجرا کنید و برای هر مرحله تغییرات حاصل در آرایه را نشان دهید .

$$(12, 6, 9, 7, 12, 15, 3, 10)$$

(۱۲، ۶، ۹، ۷، ۱۲، ۱۵، ۳، ۱۰)
(۱۲، ۶، ۹، ۷) (۱۲، ۱۵، ۳، ۱۰)
(۱۲، ۶) (۷، ۹) (۱۲، ۱۵) (۳، ۱۰)
(۱۲) (۶) (۹)(۷) (۱۲) (۱۵) (۳) (۱۰)
(۶، ۱۲) (۷، ۹) (۱۲، ۱۵) (۳، ۱۰)
(۶، ۷، ۹، ۱۲) (۳، ۱۰، ۱۲، ۱۵)
(۳، ۶، ۷، ۹، ۱۰،۱۲، ۱۲، ۱۵)

۲. [۱۵] با توجه به مبحث تقسیم و حل ارائه شده در کلاس، میدانیم که ضرب دو چند جمله ای با درجه n در زمان $O(n^2)$ قابل انجام می باشد. نقش بازنویسی حاصل عبارت $ad + cb$ به صورت $(a + b)(c + d) - ac - bd$ را در بهبود پیچیدگی محاسباتی الگوریتم ضرب چند جمله ای توضیح داده و همچنین رابطه بازگشتی، پیچیدگی **محاسباتی** و پیچیدگی **حافظه ای** آنرا با استفاده از Master Theorem به دست بیاورید .

If we write ad + bc as (a + b)(c + d) − ac − bd, then we only need to compute three integer multiplications of size n/2, namely ac, bd, and (a + c)(c + d). This is advantageous since we replace one multiplication with additions and subtractions, which are less expensive operations. The divide step will now take time $\Theta(n)$, since we need to calculate a + c and c + d, and the recombining step will still take $\Theta(n)$. This leads to the recurrence relation $T(n) = 3T(n/2) + \Theta(n)$. We have a = 3, b = 2, and f(n) = n

۳. [۱۵] آرایه ای به طول ۵۱۱ داریم . میخاهیم با استفاده از binary search عدد ۱۱۰ را در این آرایه جستجو کنیم . اگر این عدد در آرایه موجود نباشید چه تعداد مقایسه در اجرا الگوریتم رخ داده است ؟

کدام ایندکس های آرایه مورد مقایسه قرار گرفتند ؟

در صورتی که الگوریتم اجرایی linear search باشد تعداد مقایسه ها چه تفاوتی می کند ؟

You will only need to compare 110 with 9 elements of the array, because 10 is roughly the binary logarithm of 511. For example, if 110 is smaller than all the numbers in the array, you will first compare it with the element with index 255 (starting from 0), then the element with index 127, then with elements with indices 63, 31, 15, 7, 3, 1 and 0 before you determine that 110 is smaller than all numbers in the array.

if you want to linear search You will have to look at all the numbers in the array.

۴. [۲۰] درستی یا نادرستی عبارات زیر را با علامت (✓) یا (✗) مشخص کنید. دلیل خود را در نقطه چین زیر هر عبارت توضیح دهید.

(a) ✗ The following array is a max heap: [10, 3, 5, 1, 4, 2]

False. The element 3 is smaller than its child 4, violating the max heap property.

(b) ✗ In max-heaps, the operations insert, find-max, and find min all take O(log n) time.

False. The minimum can be any of the nodes without children. There are n/2 such nodes, so it would take $\Theta(n)$ time to find it in the worst case.

(c) ✓ stack is the most suitable data structures if you only need to implement recursion in a programming language

True. You put the function and its parameters values on the stack when you make recursive call, and you remove the top element of the stack when you go out of the recursive call. Stack is LIFO - last in

(d) ✓ tree is the most suitable data structure if you need to store the directory structure on your hard drive

true: The directory structure is a tree, so it is good to store it as a tree data structure.

۵. [۱۰] چگونه با استفاده از یک $priopityQueue$ یک $stack$ پیاده سازی کنیم؟

In priority queue, we assign priority to the elements that are being pushed. A stack requires elements to be processed in Last in First Out manner. The idea is to associate a count that determines when it was pushed. This count works as a key for the priority queue. The count is increased for every push/enqueue. So, assuming a max-queue, the last item always has the highest count/key, hence it will always be the first out.

So the implementation of stack uses a priority queue of pairs, with the first element serving as the key.

pair<int, int> (key, value)

۶. [۱۵] توضیح دهید قطعه کد روبرو چه کاری انجام میدهد .

```
void function()
Node * current ← head
Node * pre ← Null
Node * next ← Null
while current ≠ NULL do
    next ← current− > next
    current− > next ← prev
    prev ← current
    current ← next
end
head ← prev
```

سپس الگوریتم را بر روی $LinkList$ مقابل به صورت مرحله به مرحله اجرا کنید و خروجی را نمایش دهید .

$1- > 2- > 3- > null$

This function reverce given link list .

1.Initialize three pointers prev as NULL, curr as head and next as NULL.

2.Iterate trough the linked list. In loop, do following.

Before changing next of current,

store next node

next = curr->next

Now change next of current

This is where actual reversing happens

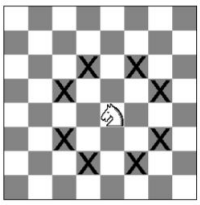
curr->next = prev

Move prev and curr one step forward

prev = curr

curr = next

۷. [۱۵] یک اسب را در صفحه شطرنج $n * n$ در نظر بگیرید . خانه هایی که اسب میتواند به آن ها برود در تصویر زیر قایل مشاهده است . یک الگوریتم برای محاسبه تعداد راه هایی که در آن می توانید با $k >= 0$ حرکت از خانه (is, js) به (is, js) برود طراحی کنید . همچنین پیچیدگی **محاسباتی** و پیچیدگی **حافظه ای** آنرا به دست بیاورید .



We can calculate the solution to this problem by noticing that if we knew the number of ways to get to every square on the board in $k1$ moves from the starting location, we could easily calculate the number of ways to get to a square in k moves by simply summing over the atmost 8 squares that the knight could move from. Each way to get to the predecessor in $k1$ moves contributes one way to get to the square in k moves.

Our DP matrix will be nnk. We initialize for all $k = 0$ the number of ways to get to that square in 0 moves. Namely, W ays(a, b, 0) = 1 if a = is AND b = js and zero otherwise. We can build up our DP matrix for each 0 < i k from these values. For each value of i = 1 . . . k, and for each square (a, b) on the n × n board, we set the value of W ays(a, b, i) = P (u,v) neighbors(a,b) W ays(u, v, i − 1). The neighbors of a cell are simply those that follow the condition given above for the legal knight's moves. At the end, we look at W ays(it , jt , k) to find the number of ways to get to (it , jt) from (is, js) in exactly k moves.

Notice here a trend common to DP problems. In order to solve how many ways there are to get from a certain cell to another cell in k moves, we actually solve more than asked for. We figure out how many ways there are to get from the start cell to every cell in k moves. While it seems like we do more computation, this actually makes solving the next layer more efficient. The running time of this algorithm will be proportional to the number of cells in the matrix, since each cell takes a constant amount of time to calculate. Thus, the running time is Θ(n 2k). The amount of space is the same at Θ(n 2k). Notice, however, that to calculate the matrix for a given value of i we only use the values at i − 1. So, at any time, we don't need to store the entire n 2k matrix, we only need two levels of it. If we delete levels (in k) as we finish using them, we only need space $\theta(n^2)$.