3. در کد زیر درست است که دو حلقه ی تو در تو داریم ولی دقت کنید که در حلقه ی داخلی با توجه به صورت مسأله که ذکر کرده است ورودی این سؤال یک جایگشت است از اعداد 0 تا n – 1 است ، با هر عمل swap نهایتاً جای یک خانه درست می‌شود (چرا؟) بنابراین این الگوریتم نهایتاً n بار انجام می‌شود و هر بار جای یک خانه درست می‌شود و الگوریمت از O(n) است

4. ابتدا عنصر میانی را بررسی میکنیم. چون آرایه مرتب شده است شرط array[i] = i را بررسی میکنیم (بررسی میکنیم که آیا عنصر میانی مقدارش با شماره ی خانه برابر است یا خیر)
اگر برابر بود که بدین معناست که جواب را یافته ایم. اگر برابر نبود دو حالت داریم
در حالت اول array[i] > i است. یعنی مقدار خانه از شماره ی خانه بیشتر است. با توجه به اینکه آرایه تنها دارای اعداد صحیح است و آرایه مرتب شده است بنابراین خانه ی array[j] و j > i مقدارش حداقل j – i تا بیشتر از array[i] است. یعنی درواقع با توجه به شرط مرتب بودن آرایه میدانیم جواب (اگر وجود داشته باشد) در قسمت سمت چپ آرایه (نسبت به خانه ی i) است.

در حالت دوم اگر array[i] < i باشد به همان شکل بالا جواب حتماً در سمت راست آرایه نسبت به i است.
بنابر این هر دفعه با بررسی خانه ی وسط میتوان فهمید که سمت راست آرایه را بررسی کنیم یا سمت چپ و هر بار با نصف کردن مسأله میتوان مسأله را در O(logn) حل کرد

5. مطمئناً اگر عدد غالب وجود داشته باشد آن عدد، همان عددی است که در خانه ی میانی آرایه وجود دارد (چرا؟)
بنابراین ابتدا عنصر میانی را مشاهده میکنیم و اسم آن را a مینامیم.
حال ابتدا باید پیدا کنیم که عنصر a در کدام خانه برای اولین بار مشاهده شده است. (آرایه مرتب شده است)
برای این کار از یک رویکرد شبیه به باینری سرچ استفاده میکنیم بدین شکل که ابتدا در نیمه ی سمت چپ آرایه، عنصر میانی را مشاهده میکنیم. اگر عنصر مشاهده شده برابر a بود دوباره در نیمه ی سمت چپ آن، عنصر میانی را مشاهده میکنیم و این عمل را مرتبا تکرار میکنیم.
اگر عنصر مشاهده شده کمتر از a باشد، از نیمه ی سمت راست، عنصر میانی را مشاهده میکنیم و بررسی میکنیم و این عمل را تکرار میکنیم تا نهایتاً به اولین خانه‌ای که عنصر a در آن تکرار شده است برسیم.
به همین ترتیب آخرین خانه‌ای که عنصر a در آن وجود دارد را نیز باید پیدا کرد (روش این کار را برای خود توضیح دهید :)
نهایتاً با پیدا کردن ابتدا و انتهای تکرار a میتوان با استفاده از یک تفریق تعداد تکرار a را به دست آورد و غالب بودن آن را بررسی کرد.

6. عنصر مینیمال : ابتدا عنصر میانی را با دو عنصر کناری خود مقایسه میکنیم. اگر از هر دو کمتر باشد بنابراین یک عنصر مینیمال است. در غیر این صورت :
فرض کنیم که از عنصر سمت راست خود بزرگ‌تر باشد. بنابر این جواب حتماً در قسمت راست است (چرا؟ برهان خلف. فرض کنیم عنصر میانی از عنصر سمت راستی خود بزرگ‌تر باشد ولی جواب در نیمه ی راست

آرایه وجود نداشته باشد. بنابراین عنصر سمت راستی عنصر میانی را a مینامیم. عنصر سمت راستی a نمیتواند بزرگ‌تر از a باشد زیرا در این صورت a مینیمال میشود. پس عنصر سمت راست a از a کوچکتر است. به همین ترتیب همین شرایط برای عنصر سمت راستی بعدی نیز صدق میکند و نهایتاً به عنصر انتهایی میرسیم که از عنصر سمت چپ خود کوچکتر است که باعث می‌شود عنصر مینیمال شود)

اگر عنصر میانی از عنصر سمت چپ خود بزرگ‌تر باشد. در این حالت مانند قبل حتماً یک جواب در نیمه سمت چپ وجود دارد.
بنابراین هر دفعه با نصف کردن مسأله میتوان مسأله را در logn حل کرد.

نکته: در حل‌ها سعی شد تنها ایده‌ها ارائه شود و از طرح جزئیات پرهیز شود. شما باید هنگام ارائه ی الگوریتم خود به جزئیات نیز توجه کنید. مثلاً هنگام بررسی زیر مسأله ها با نصف کردن زیر مسأله در هر کدام از مسأله های 5 یا 6 طول زیر مسأله ممکن است 0 یا 1 شود و مثلاً عنصر سمت راست یا سمت چپ میانی برای مقایسه وجود نداشته باشد.
یا شرایط خاصی مانند این پیش بیاید
شما باید در الگوریتم های خود این شرایط را نیز لحاظ کنید و آن را نیز توضیح دهید
یا برای مثال در سؤال 6 ممکن است عنصر مینیمال وجود نداشته باشد (آرایه ای که تمام اعضای آن مانند هم باشند)
یا در سؤال 5 ممکن است آرایه عنصر غالب نداشته باشد
یا در سؤال 4 ممکن است هیچ عضوی با شرایط بالا وجود نداشته باشد
الگوریتم های شما باید این حالات را نیز بتوانند تشخیص دهند