《 수학I 수행평가 주제 》

하이에듀

주제	이진로그를 활용한 이진검색 및 관련 알고리즘
요약	이진 검색에 대한 기본적인 개념을 살펴보고, 그 구현을 어떻게 하는지 의사 코드를 통해 공부해보면 좋을 것 같습니다. 그리고 이진 검색의 실행 시간에 대해 살펴보면 좋을 것 같습니다. 실행 시간에 대한 내용에 수학1의 로그에 대한 내용이 들어가니, 이를 중점적으로 확인해주시면 될 것 같습니다.

자료 1. 이진 검색

이진 검색은 정렬된 리스트에서 원하는 항목을 찾기에 효율적인 알고리즘입니다. 이 검색법은 후보 범위가 한 항목으로 좁아질 때까지 찾고자 하는 항목을 포함하고 있는 리스트를 반으로 나누는 과정을 계속 반복합니다. 이전에 추측 게임에서 이진 검색을 사용해 보았습니다.

이진 검색을 가장 많이 사용하는 경우는 배열에서 어떤 항목을 찾아야 할 때입니다. 예를 들어 티코(Tycho)-2 항성 목록은 우리 은하계에서 가장 밝은 별 2,539,913개의 정보를 담고 있습니다. 별의 이름을 검색하여 항성 목록에서 어떤 별을 찾고 싶다고 가정해 봅시다. 프로그램이 **선형 검색(linear search)**을 이용해서 별 카탈로그의 첫 번째부터 차례대로 별을 찾는다면, 최악의 경우에 2,539,913개의 별을 모두 검사해야 할 수도 있습니다. 카탈로그가 별 이름에 따라 알파벳 순서로 정렬되어 있다면, 최악의 경우라도**이진 검색**으로는 22개의 별만 탐색하면 됩니다.
앞으로 볼 개념이해하기 글에서는 알고리즘을 자세하게 묘사하는 방법, 알고리즘을 자바스크립트로 구현하는 방법 그리고 알고리즘의 효율성을 분석하는 방법에 대해 설명해 보겠습니다.

이진 검색 묘사하기

사람들에게 어떤 알고리즘을 설명할 때에 따라서는 꼭 완벽하게 설명하지 않아도 됩니다. 케이크 조리법에서 요리할 때 달걀을 냉장고에서 꺼내기 위해 어떻게 냉장고를 열지, 달걀을 어떻게 깨는지 같은 자잘한 사항은 생략합니다. 이 정도는 알 것이라고 가정하는 것이죠. 사람들은 생략된 내용을 어떻게 채울지 직관적으로 알지만, 컴퓨터 프로그램은 그렇지 않습니다. 그러므로 컴퓨터 알고리즘은 완벽하게 작성해야 합니다.
프로그래밍 언어로 알고리즘을 구현하기 위해서는 알고리즘을 자세하게 이해해야 합니다. 문제에 입력값은 무엇인지, 출력값은 무엇인지, 어떤 변수를 만들어야 하고 초기값은 어떻게 설정해야 하는지, 또 마지막 출력값을 계산하기 위해서 어떤 중간 과정이 필요한지, 이

모든 과정이 단순 반복문을 이용해 쓸 수 있는 명령일까요?
이진 검색을 자세하게 서술하는 법을 살펴봅시다. 이진 검색의 주 원리는 현재 머물러있는 합리적 추측 범위를 계속 파악하는 것입니다. 추측 게임을 한다고 가정하고 1부터 100까지의 숫자 중 하나를 골랐습니다. 만일 25라고 추측하면 맞추고자 하는 숫자는 그것보다 큰 수라고 얘기해주고, 또 81으로 추측할 경우 숫자가 그보다 작다고 말해 준다면 정답은 26부터 80까지의 범위에 있다고 추측하는 것이 합리적일 것입니다. 아래 수직선에서 빨간 부분은 합리적 추측값이고 까만 부분은 배제되는 부분입니다.

매 회마다 추측값 한 개를 선택해서 바로 전에 나온 합리적 추측 범위를 둘로 나누어 비슷한 크기의 범위가 두 개 나오도록 합니다. 추측이 틀리면 값이 너무 높거나 너무 낮다고 말할 것이고, 이에 따라 합리적 추측값의 절반 범위를 제외할 수 있습니다. 예를 들어 합리적인 추측값의 범위가 26에서 80 이면, 이 절반인 (26+80)/2, 53 을 선택할 것입니다. 그리고 53이 정답보다 크다면 53에서 80까지 범위는 제외할 수 있고, 26에서 52 범위가 다시 합리적 추측 범위가 되어 범위의 크기를 절반으로 줄이는 겁니다.

추측 게임에서는 변수 몇 개를 사용하여 합리적 추측 범위를 여러 개 추적할 수 있습니다. 변수 min을 현재 최소 합리적 추측값이라고 하고 max를 현재 최대 합리적 추측값이라고 합시다. 문제의 *입력값*은 상대방이 생각할 수 있는 가장 큰 숫자인 n입니다. 답이 될 수 있는 가장 작은 값은 1이라고 가정하지만, 가능한 가장 작은 값을 두 번째 입력값으로 취하도록 알고리즘을 수정하기는 쉽습니다.

다음은 추측 게임에서 이진 검색을 사용하는 방법을 단계별로 나타낸 것입니다:

1. *min* = 1, *max* = *n*으로 둡니다.
2. *max*와 *min*의 평균을 구하되, 정수가 되도록 내림합니다.
3. 추측이 맞으면 끝냅니다. 숫자를 찾았습니다!
4. 추측값이 너무 작으면 *min*을 추측값보다 1 크게 설정합니다.
5. 추측값이 너무 크면 *max*를 1 작게 설정합니다.
6. 2 단계로 돌아갑니다.

알고리즘의 입력값과 출력값을 분명하게 설정하고 "평균을 구함 "이나 "끝냅니다" 같은 명령이 정확히 의미하는 바를 명시하면 훨씬 더 명확하게 설명할 수 있습니다. 하지만 지금은 이것으로 충분합니다.

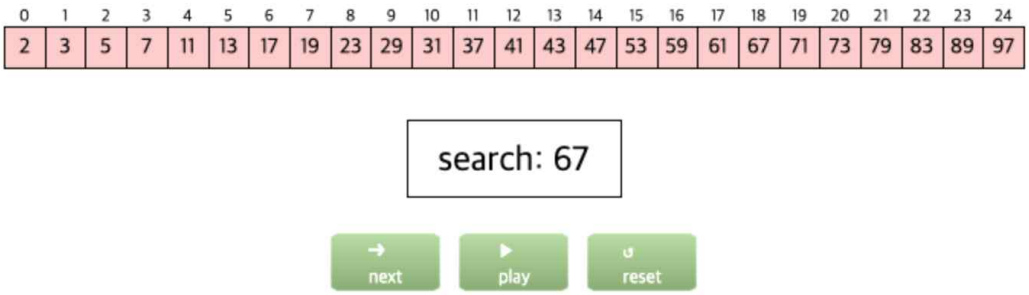
자료 2. 이진검색의 구현과 의사코드

정렬된 배열에서 이진 검색을 사용하는 방법을 배워 봅시다. 자바스크립트와 다른 수많은 프로그래밍 언어는 이미 주어진 요소가 배열 안에 있는지, 만일 있다면 그 위치를 알려주

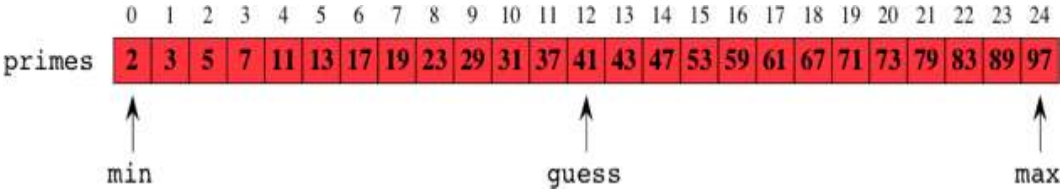
는 메서드를 가지고 있습니다. 하지만 이런 메소드를 더 잘 이해할 수 있도록 직접 구현해 봅시다. 다음은 25개 소수가 차례대로 저장된 JavaScript 배열입니다.

```
var primes = [2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41,
43, 47, 53, 59, 61, 67, 71, 73, 79, 83, 89, 97];
```

예를 들어 숫자 67이 소수인지 알고 싶다고 합시다. 67이 배열 안에 있다면 67은 소수입니다.
또 67보다 작은 소수는 몇 개일지 궁금할 수 있습니다. 배열에서 67의 위치를 알아내면 이 정보로 67보다 작은 소수가 몇 개인지를 알아낼 수 있습니다.
어떤 요소의 배열 내 위치를 인덱스라고 합니다. 배열의 인덱스는 0부터 시작해서 하나씩 커집니다. 인덱스 0에 있는 요소는 배열 내 첫 번째 요소입니다. 어떤 요소가 인덱스 3에 있다면 그 앞에 세 개의 요소가 있음을 의미합니다.
아래 예시를 살펴봅시다. 왼쪽에서 오른쪽으로 소수의 배열을 읽어내려가면 (핑크 네모 속에 있는) 숫자 67을 배열 인덱스 18에서 찾을 수 있습니다. 이런 식으로 순서대로 숫자를 살펴보는 것을 *선형 검색*이라고 합니다.
67이 인덱스 18에 있다는 것을 확인하면 67이 소수라는 것을 확인할 수 있습니다. 또한 배열 내에 67 밑으로 18개의 요소가 있다는 것, 즉 67보다 작은 소수는 18개라는 것을 금방 알 수 있습니다.

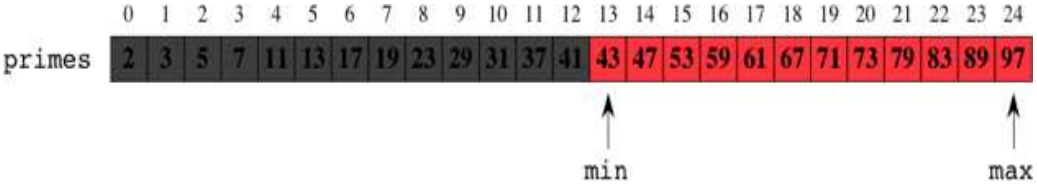


여기까지 몇 단계가 걸렸는지 보셨나요? 이진 검색이 더 효율적일 수 있습니다. primes 배열에는 25개의 소수가 포함되어 있기 때문에 배열의 인덱스 범위는 0부터 24까지입니다. 이전처럼 의사코드를 이용하여 min = 0 , max = 24로 설정하면서 시작합니다. 이진 검색에서의 첫 번째 추측값은 (0+24)/2, 즉 12번 인덱스에 있는 값입니다. primes[12]은 67인가요? 아닙니다, primes[12]는 41입니다.

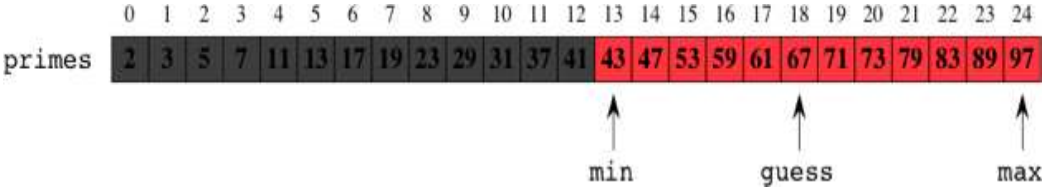


찾고자 하는인덱스값은 12보다 클까요, 작을까요? 배열 내 값은 오름 차순으로 되어 있고 41 < 67이기 때문에 67이라는 값은 인덱스 12의 오른쪽에 있을 것입니다. 즉 12보다 큰

인덱스를 찾아봐야 한다는 것입니다. min을 12+1인 13으로 업데이트하고 max는 24로 놔둡니다.



다음에 추측해야 하는 인덱스는 무엇일까요? 13과 24의 평균값은 18.5이고 배열 인덱스는 반드시 정수여야 하므로 18로 반내림합니다. 이제 primes[18]이 67이라는 것을 확인할 수 있습니다.



답을 찾았으므로 이진 검색 알고리즘은 여기서 멈춥니다. 선형 검색에서는 19번의 추측이 이루어진 반면 여기서는 두 번만에 끝났습니다.

의사코드(pseudo-code)

한 가지 예를 들어 이진 검색 알고리즘을 글로 설명했습니다. 글로 나타내는 방법도 좋지만 글로 표현하다 보면 내용에 차이가 생길 수도 있습니다. 글은 너무 짧거나 너무 길 수도 있고 또한 가장 중요한 것은 항상 정확할 수는 없다는 것입니다. 여기서 JavaScript나 Python과 같은 프로그래밍 언어로 이진 검색을 설명할 수도 있지만, 프로그램에는 엄청나게 많은 정보가 들어있습니다. 프로그래밍 언어에 따른 요구사항을 채울 부분과 또 잘못된 데이터, 사용자 에러, 시스템 에러 등에도 대처해야 합니다. 그러므로 코드만 공부하게 되면 코드에 적용된 알고리즘을 이해하기가 힘들어질 수 있습니다. 이런 점으로 인해 본문에서는 종종 프로그래밍 언어의 특징들과 글을 혼합하여 표현하는 의사코드로 알고리즘을 설명합니다.

다음은 배열 안에서 검색이 가능하도록 수정된 이진 검색 의사코드입니다. 입력값은 array라고 부르는 배열, array의 요소의 개수 n, 검색 대상의 수 target입니다. 결과값은 array 속 target의 인덱스 값입니다.

1. min = 0 이고 max = n-1이라고 합니다.
2. guess의 값은 max와 min의 평균값을 정수가 되도록 버림한 값입니다.
3. array[guess]의 값이 target과 같다면 검색을 멈춥니다. 타겟을 찾았습니다. guess를 결과값으로 반환합니다.
4. 추측값이 더 작다면 즉, array[guess] < target이라면, min = guess + 1으로 바꿉니다.
5. 아니면 추측값이 더 큽니다. 그러면 max = guess - 1로 바꿉니다.
6. 2 단계로 돌아갑니다.

의사코드의 구현

이제부터는 상황에 따라 말로 된 설명, 의사코드, 그리고 자바스크립트 언어를 번갈아서

사용할 것입니다. 프로그래머라면 의사코드를 이해하고 이를 원하는 언어로 바꿀 수 있어야 합니다. 그러므로 여기서 JavaScript를 사용해도 다른 언어로 의사코드를 쉽게 구현할 수 있어야 합니다.

의사코드를 어떻게 해야 자바스크립트 프로그램으로 바꿀 수 있을까요? 먼저 함수를 생성해야 합니다. 왜냐하면, 입력을 받아서 값을 반환하는 코드를 작성해야 하고, 또, 이 코드를 다른 곳에서 다른 입력값을 받아서 써야 하기 때문입니다. 함수(binarySearch 라고 합시다)의 매개변수로는 배열과 찾고자 하는 대상 값이 있고 반환할 값은 찾고자 하는 값의 위치를 나타내는 인덱스입니다.

이제 함수의 본체 코드로 들어가서 어떻게 구현할지 살펴봅시다. 6단계는 2단계로 돌아가라고 합니다. 반복문처럼 보이네요. for 문 또는 while 문 중 무엇을 써야 할까요? for 문을 꼭 사용하고 싶다면 그럴 수도 있지만 이진 검색에서 추측하는 인덱스는 for 문과 같이 순서대로 커지지 않습니다. 처음에는 인덱스 12번을 추측하고 그 후에는 계산하여 18번으로 추측했습니다. 그래서 while 문이 더 좋은 선택입니다.

추측 게임 알고리즘을 짤 때 만든 의사 코드에서는 중요하지 않았지만, 배열의 이진 검색에서는 한 가지 중요한 단계를 빠뜨렸습니다. 만약 찾고 싶은 숫자가 배열에 *없다면* 어떤 일이 일어날까요? 이 경우에는 binarySearch 함수가 돌려주어야 하는 인덱스값을 먼저 결정합니다. 이 값은 배열에서 인덱스가 될 수 없는 숫자여야 합니다. -1은 배열 내 인덱스가 될 수 없으므로 -1을 사용하기로 합시다. (사실 아무 음수 모두 괜찮습니다.)

추측을 더 이상 할 수 없다면 검색 대상 값이 배열에 존재하지 않는 것입니다. 이 예제의 primes 배열에서 검색 대상 10을 찾는다고 해봅시다. 그 대상이 배열 안에 존재한다면, 인덱스 값이 각각 3과 4인 7과 11사이에 있을 것입니다. binarySearch 함수가 실행되는 동안 min과 max의 인덱스 값 변화를 보다보면, min 값이 3이 되고 max 값이 4가 되는 경우가 생깁니다. 그러면 추측값은 인덱스 3이 되는데 ((3 + 4) / 2 는 3.5로 버림하면 3이기 때문), primes[3]은 10보다 작고, min값이 4가 됩니다. min과 max가 모두 인덱스 4이기 때문에 다음 추측값은 인덱스 4가 되고 primes[4]는 10보다 큽니다. 그에 따라 max 값이 3이 됩니다. min이 4이고 max가 3이라는 것은 무슨 뜻일까요? 가능한 추측값이 최소 4이고 최대 3이라는 뜻입니다. 이 조건을 성립하는 수는 없습니다! 따라서, 검색 대상인 10이 primes 배열에 없다고 결론내릴 수 있고, binarySearch 함수가 -1을 반환합니다. 일반적으로, max가 min보다 작아진다면, 검색 대상이 정렬된 배열에 존재하지 않는다는 것을 알 수 있습니다. 검색 대상이 배열에 없는 경우를 처리하는 이진 검색 의사코드는 다음과 같습니다.

1. min = 0 이고 max = n-1이라고 합니다.
2. max < min이면 검색을 멈춥니다: target이 array에 존재하지 않습니다. -1을 반환합니다.
3. guess 의 값은 max 와 min 의 평균값을 정수가 되도록 버림 한 값입니다.
4. array[guess]의 값이 target과 같다면 검색을 멈춥니다. 타겟을 찾았습니다. guess를 결과값으로 반환합니다.
5. 추측값이 더 작았다면 즉, array[guess] < target이라면, min = guess + 1으로 바꿉니다.
6. 다른 경우로는 추측값이 더 클 수 있습니다. 그러면 max = guess - 1로 바꿉니

다.

7. 2단계로 돌아갑니다.

이제까지 함께 의사코드를 살펴보았습니다. 이제 스스로 이진 검색을 구현해 보세요. 의사 코드 부분을 복습해도 괜찮습니다. 오히려 복습해보는 편이 좋습니다. 그러면 의사 코드를 프로그램으로 바꾸는 과정을 더욱 더 잘 이해할 수 있을 것입니다.

자료 3. 이진검색의 실행 시간

n개의 요소가 있는 배열에서 선형 탐색으로 탐색을 하면 최대 n번의 탐색을 거쳐야 합니다. 아마 직관적으로 이진 탐색으로 훨씬 빨리 탐색을 할 수 있을거라는 생각이 들겁니다. 배열의 길이가 증가할수록 선형 탐색과 이진 탐색의 속도 격차는 더더욱 벌어집니다. 이진 탐색의 최대 탐색수를 알아보고 이를 분석하는 법을 배워 봅시다.

여기서 핵심 개념은 다음과 같습니다. 이진 탐색에서는 기준값과 배열의 정중앙에 있는 데이터를 비교하여 맞지 않을 경우 배열에 있는 기준값과 일치하지 않는 다른 쪽에 있는 값들은 모두 버립니다. 따라서 남은 값들에 대한 탐색을 다시 시작할 때 옳은 값을 찾을 확률은 이전 탐색에 비해 최대 두 배가 올라갑니다. 예를 들어 어떤 32개의 데이터가 있다고 가정해봅시다. 정렬된 데이터 중간의 어떤 값 하나를 비교해 보았더니 찾는 값보다 더 커서 이 값보다 큰 값들을 모두 버렸습니다. 그럼 남은 16개의 더 작은 값 중에서 탐색을 다시 시작하여 원하는 값을 찾게 됩니다. 이처럼 이진 탐색은 비교값이 틀릴 때마다 값의 범위를 반으로 줄입니다.

만일 길이가 8인 배열에서 탐색을 시작하여 비교값이 틀리면 다시 4개의 값에서 탐색하고 다음으로 2, 1 순으로 줄여나갑니다. 탐색하는 배열 안에 값이 한 개만 남아있으면 비교값과 일치 여부에 상관없이 탐색은 종료됩니다. 따라서 길이가 8인 배열의 경우 이진 탐색은 최대 네 번 탐색을 합니다.

그렇다면 배열에 값이 16개 있을 경우에는 어떻게 될까요? 첫 번째 탐색이 최소 8개의 값을 제외시켜서 남는 8개의 값에 대해 탐색을 한다고 생각했다면 이해를 올바르게 하고 있는 것입니다. 따라서 16개의 값을 가지는 배열에서는 탐색을 최대 5번만 하면 됩니다.

지금쯤이면 슬슬 규칙이 보이기 시작할 겁니다. 배열이 두 배 커질때마다 최대 탐색 횟수는 한 번 늘어납니다. 길이가 n인 배열에 최대 m 번 만큼의 탐색이 필요하다고 가정해봅시다. 그렇다면 길이가 2n인 배열에서는 첫 번째 탐색을 한 다음 탐색 범위가 n 으로 절반이 되므로 최대 m 번의 탐색만 하면 원하는 값을 반드시 찾게됩니다. 따라서 이 경우 최대 m+1 번의 탐색이 필요합니다.

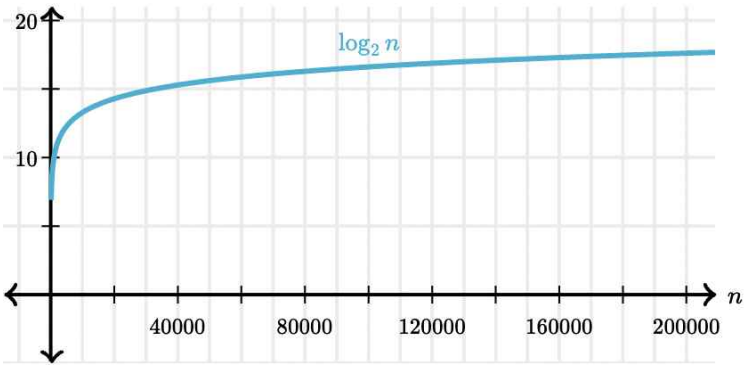
길이가 n인 배열의 경우에 최악의 경우 추측 횟수는 n에서 시작해 1에 이르기까지 반복적으로 반으로 나누는 횟수 더하기 1이라고 할 수 있습니다. 하지만 이것을 말로 쓰기엔 불편합니다.

다행히 n에서 시작해 1에 이르기까지 반복적으로 반으로 나누는 횟수를 뜻하는 수학 함수가 존재합니다. 바로 **2를 밑으로 하는 n의 로그**입니다. 이는 보통 log2n이라고 쓰지만 컴퓨터 과학에서는 lg$n$이라고 쓸 때도 있습니다. (로그에 대해 더 알고 싶다면 여기를 살펴보세요.)

아래 표는 2를 밑으로 하는 n의 로그함수 값 중 일부입니다:

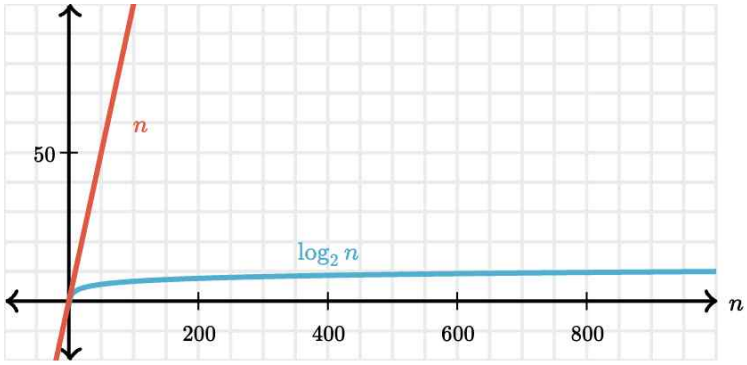
n	$\log_2 n$
1	0
2	1
4	2
8	3
16	4
32	5
64	6
128	7
256	8
512	9
1024	10

이 표를 그래프로 나타낼 수도 있습니다:



로그함수는 매우 천천히 증가합니다. 로그함수는 매우 빠르게 증가하는 지수함수와 역의 관계에 있습니다. 로그함수가 log2*n*=*x* 라면 그에 대응하는 지수함수는 *n*=2^x 입니다.
n이 2의 제곱일 때 이진 검색 알고리즘의 실행 시간을 계산하는 것은 간단합니다. n이 128이라면, 이진 검색은 최대 8($\log_2 128$ +1)번의 추측만 있으면 됩니다.
n이 2의 제곱이 아니라면 어떨까요? 그런 경우 n보다 낮은 가장 가까운 2의 제곱인 수를 고르면 됩니다. 길이가 1000인 배열이 있다면, n보다 낮은 가장 가까운 2의 제곱은 512(2^9)입니다. 따라서 log2 1000은 9와 10 사이라고 추측할 수 있고, 실제로 계산해보면 9.97 정도입니다. 여기에 1을 더하면 10.97이고, 소수가 나오는 경우 정수로 내림합니다. 따라서 1000개의 요소를 가진 배열을 이진 검색할 때는 최대 10번의 추측만 필요합니다.

항성 2,539,913개를 가진 Tycho-2 항성 목록의 경우 가장 가까운 2의 제곱은 2^21 (2,097,152)이므로, 최대 22번의 추측이 필요합니다. 선형 검색보다 *훨씬* 낫습니다.
다음은 n과 log2 n를 비교한 그래프입니다:



https://ko.khanacademy.org/computing/computer-science/algorithms/binary-search/a/running-time-of-binary-search