```r
res <- function(theta, r, s) {
  if (theta < 0) {
    return (2 - pnorm(r-theta, mean = 0, sd = 1) - pnorm(s-theta, mean = 0, sd = 1) )
  }
  else if (theta == 0) {
    return(pnorm(r, 0, 1) + 1 - pnorm(s, 0, 1))
  }
  else if (theta > 0 ){
    return(pnorm(r - theta, 0, 1) + pnorm(s - theta, 0, 1))
  }
}

x <- seq(-3, 3, length = 200)

res1 <- function(x){ return(res(x, -1, 1)) }
```

theta, r, s의 값에 따라 다르게 확률을 계산하는 함수 res를 정의하고, -3~3 사이에 있는 200개의 실수로 시퀀스를 정의하여 x에 할당합니다.

이제 res1이라는 함수를 위에서 정의한 함수로 다시 만들어주고, x에 for문과 sapply를 적용하여 계산하여 각 방법에 시간이 얼마나 걸리는지 확인봅니다.

```r
# when use for loop
start_time1 <- Sys.time()

for_res = c()
for (i in x){
  tmp = res1(i)
  for_res = c(for_res, tmp)
}

end_time1 <- Sys.time()
elapsed_time1 <- as.numeric(difftime(time1 = end_time1, time2 = start_time1,
                                     units = 'secs'))
```

for 문을 사용하여 벡터 x의 각 원소에 대하여 계산을 수행하고, 총 걸린 시간을 elapsed_time1에 저장해둡니다.

다음으로 sapply를 사용하여 계산을 수행해보겠습니다.

```r
# when use sapply
start_time2 <- Sys.time()

apply_res = sapply(x, res1)

end_time2 <- Sys.time()
elapsed_time2 <- as.numeric(difftime(time1 = end_time2, time2 = start_time2,
                                     units = 'secs'))
```

for문에서와 같이 총 걸린 시간을 elapsed_time2에 저장합니다.

```
43  cat('elapsed time : ', sprintf("%.3f", elapsed_time1), 'sec', sep = "")
44  cat('elapsed time : ', sprintf("%.3f", elapsed_time2), 'sec', sep = "")
```

걸린 시간을 확인해보면,

```
> cat('elapsed time : ', sprintf("%.3f", elapsed_time1), 'sec', sep = "")
elapsed time : 0.013sec> cat('elapsed time : ', sprintf("%.3f", elapsed_time2), 'sec', sep = "")
elapsed time : 0.001sec
```

for문을 사용한 경우 0.013초가, sapply를 사용한 경우 0.001초가 걸렸습니다.

for문을 사용했을 때보다 sapply를 사용했을 때 계산 속도가 더 빨라졌습니다.